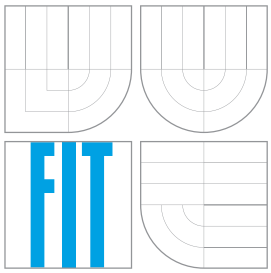
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VÝSTAVBA A PROGRAMOVÁNÍ CLUSTERU O NÍZKÉM PŘÍKONU

DEVELOPMENT AND PROGRAMMING OF LOW POWER CLUSTER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAL HRADECKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JIŘÍ JAROŠ, Ph.D.

BRNO 2016

Zadání diplomové práce/17961/2015/xhrade08

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Hradecký Michal, Bc.**
Obor: Počítačové a vestavěné systémy
Téma: **Výstavba a programování clusteru o nízkém příkonu**
Development and Programming of Low Power Cluster
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s architekturou procesorů ARM a dostupnými vývojovými kity.
2. Seznamte se s výstavbou a administrací linuxových clusterů.
3. Prostudujte dostupné softwarové vybavení a navrhněte jednoduchý cluster složený z kitů na bázi procesoru ARM.
4. Navrhněte a implementujte sadu mikrotestů s cílem vyhodnotit základní výkonnostní parametry navrženého clusteru.
5. Zvolte vhodný problém, na němž budete demonstrovat efektivní využití navrženého clusteru.
6. Zvolený problém implementujte a otestujte.
7. Porovnejte výkonnost a spotřebu s klasickým superpočítačem.
8. Diskutujte přínos Vaší práce.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 4 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Jaroš Jiří, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 25. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Projekt se zabývá výstavbou a programováním nízko-příkonového clusteru složeného z kitů Hardkernel Odroid XU4 založených na čipech ARM Cortex A15 a Cortex A7. Cílem bylo navrhnout jednoduchý cluster složený z několika kitů a vytvořit pro něj sadu testů, na nichž by šlo otestovat základní výkonnostní parametry a spotřebu. K testování byly použity zejména benchmarky HPL, Stream a různé testy pro rozhraní MPI. Celkový výkon clusteru složeného ze 4 kitů měřený v benchmarku HPL byl 23 GFLOP/s ve dvojité přesnosti, přičemž cluster vykazoval efektivitu výpočtu cca 0,58 GFLOP/W. Práce dále popisuje instalaci plánovače PBS Torque a frameworku pro kompilaci a správu HPC softwaru EasyBuild na 32bitové platformě ARM. Po srovnání se superpočítačem Anselm vyšlo, že Odroid cluster poskytuje přibližně stejnou efektivitu výpočtu jako velký superpočítač, ovšem za vyšší pořizovací cenu za srovnatelný výkon.

Abstract

This thesis deals with the building and programming of a low power cluster composed of Hardkernel Odroid XU4 kits based on ARM Cortex A15 and Cortex A7 chips. The goal was to design a simple cluster composed of multiple kits and run a set of benchmarks to analyze performance and power consumption. The test set consisted of HPL and Stream benchmarks and various tests for the MPI interface. The overall performance of the cluster composed of four kits in HPL benchmark was measured 23 GFLOP/s in double-precision. During this test, the cluster showed power efficiency about 0.58 GFLOP/W. The work also describes the installation of PBS Torque scheduler and HPC software build and installation framework EasyBuild on 32-bit ARM platform. The comparison with Anselm supercomputer showed that Odroid cluster is as effiecient as large supercomputer but with slightly higher price.

Klíčová slova

Odroid XU4, cluster, ARM, Cortex A15, HPC, PBS, Torque, EasyBuild, superpočítač

Keywords

Odroid XU4, cluster, ARM, Cortex A15, HPC, PBS, Torque, EasyBuild, supercomputing

Citace

HRADECKÝ, Michal. *Výstavba a programování clusteru o nízkém příkonu.* Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Jaroš Jiří.

Výstavba a programování clusteru o nízkém příkonu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jiřího Jaroše, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Hradecký
24. května 2016

Poděkování

Chtěl bych poděkovat vedoucímu své práce Ing. Jiřímu Jarošovi, Ph.D., za cenné rady a čas, který mi během tvorby práce věnoval.

This work was supported by The Ministry of Education, Youth and Sports from the Large Infrastructures for Research, Experimental Development and Innovations project „IT4Innovations National Supercomputing Center – LM2015070“.

Obsah

Kapitola 1

Úvod

Současné superpočítačové systémy pro vědecké výpočty jsou většinou stavěny na platformě x86-64. Jednotlivé procesory mají velmi dobrý jednojádrový výkon, ale jsou relativně drahé a mají velkou spotřebu. Pozornost se proto nyní soustřeďuje na procesory architektury ARM, které byly navrhovány s ohledem na použití v mobilním odvětví, takže jsou zaměřeny na malý odběr a nízkou cenu za kus. Tímto směrem se dává i tato práce.

Cílem práce bylo postavit právě takový malý superpočítačový systém, a na něm otestovat, zda je typické softwarové vybavení používané na moderních superpočítačích možné používat na platformě ARM. Současně byl kladen důraz na dokumentaci celého procesu, aby vše bylo možné zopakovat při stavbě potenciálního budoucího clusteru. Byla tedy snaha o instalaci typických vědeckých knihoven, programů a prostředků pro jejich správu včetně plánovače úloh. Po dokončení nemohlo chybět otestování clusteru z hlediska výkonu a spotřeby.

V budoucnu jej také bude možné použít pro výukové účely studentů FITu, kteří si na něm budou moci vyzkoušet odlišný styl paralelního programování bez nutnosti drahého přístupu na komerční superpočítače.

Svoji roli hraje práce také v projektu k-Wave[1], prováděném na Londýnské univerzitě UCL, který se zabývá plánováním léčby rakovinných nádorů pomocí zacíleného ultrazvuku o vysoké intenzitě neboli HIFU. Zde jsou v současné době využívány moderní grafické karty, které pomáhají urychlit celý proces simulace operací. Ty jsou ale velmi náročné na pořizovací cenu a příkon, a právě takovéto clustery by mohly být nenáročným řešením problému.

Téma jsem si vybral, protože je mi blízká oblast efektivního využití hardwaru a zvolené téma bylo zajímavou příležitostí získat zkušenosti v této oblasti. Také je to v jistém slova smyslu navázání na moji bakalářskou práci, která se zabývala využitím grafických karet k urychlení simulací.

Hned v následující kapitole jsou vysvětleny termíny potřebné k pochopení všech souvislostí. Čtenář znalý tématu může tuto kapitolu přeskočit. Kapitola 3 obsahuje seznam použitého hardwarového vybavení a popř. jeho popis. V Kapitole 4 rozebírám do podrobností proces návrhu clusteru, včetně instalace softwarového vybavení a překážek, které bylo nutné překonat. V Kapitole 5 popisuji, jak probíhalo testování a měření výpočetních vlastností hotového clusteru a srovnání výsledků s již existujícími clustery. Nakonec Kapitola 6 se zabývá efektivitou spotřeby energie.

[1] http://www.k-wave.org/

Kapitola 2

Terminologie

Tato kapitola popisuje a vymezuje podstatu řešeného problému a představuje technologie, které jsou relevantní k této práci.

Za cluster můžeme považovat počítačový systém složený z několika počítačů, kterým se v této oblasti říká uzly. Tyto počítače jsou navzájem propojeny pomocí počítačové sítě. Touto sítí může být Ethernet, ale ve větších clusterech, kde je potřeba velká rychlost spojení, jsou využívány sítě poskytující větší rychlost přenosu, např. síť InfiniBand.

Při pohledu na aktuální žebříček nejvýkonnějších 500 superpočítačů světa, zvaný TOP500[1], lze zjistit, že ve většině z nich jsou použity x86-64 procesory Intel Xeon, jak lze vidět na Obrázku 2.1. Tyto procesory jsou však energeticky velmi náročné. To je problém z několika důvodů. Kromě velkého množství spotřebované elektrické energie jde i o vyprodukované odpadní teplo, které je potřeba odvádět pryč, což dále zdražuje jejich provoz, nehledě na ekologickou stránku věci. Lze tedy pozorovat tlak na energetickou úspornost superpočítačů. Ten můžeme vidět na velkém počtu superpočítačů s procesory IBM BlueGeneQ, které pravidelně obsazují přední příčky přehledu Green500, obdoby Top500, která se zaměřuje na superpočítače s nejlepším poměrem výkon / energetická náročnost.

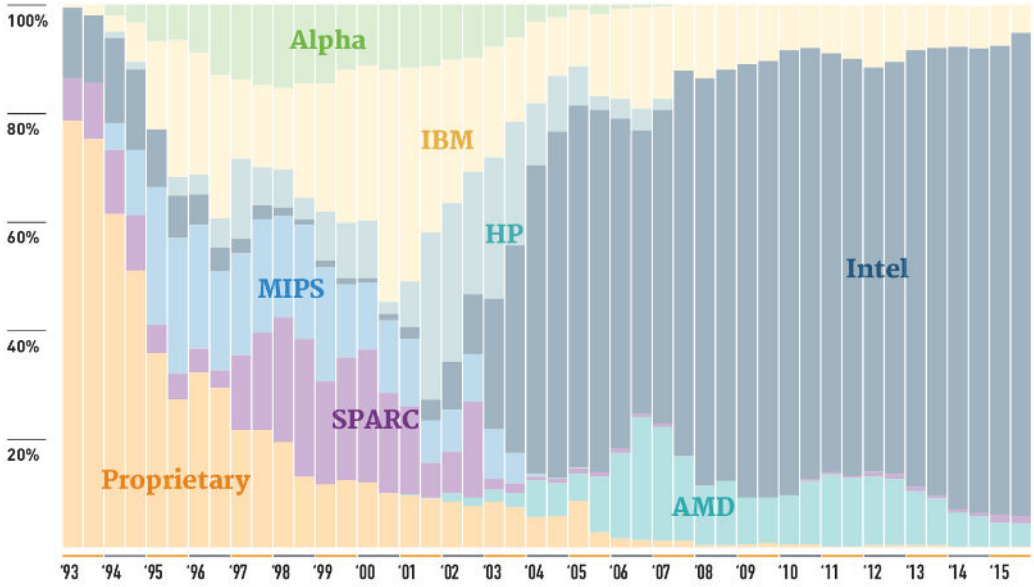
Měření výkonu

Většina vědeckých výpočtů probíhá v plovoucí řádové čárce. Pro zjištění výkonnosti procesoru (nebo algoritmu) je tedy výhodné změřit, kolik takových operací zvládne počítač vykonat za jednotku času. Efektivitu měříme v jednotkách GFLOP/s (Giga floating-point operations per second), tedy kolik miliard operací v plovoucí řádové čárce program provede za sekundu, více je lépe [9]. Pro operace v pevné řádové čárce můžeme měřit hodnotu IOP/s (integer operations per second).

High performance computing

Často zkracováno pouze jako HPC nebo česky vysoce náročné výpočty. Je to souhrnné označení pro oblast, která se zabývá využitím superpočítačů a paralelního zpracování informací pro řešení výpočetně náročných problémů, hlavně z oblasti vědy a techniky.

[1] http://www.top500.org



Obrázek 2.1: Podíl technologií čipů v TOP500 superpočítačích v čase[18].

2.1 ARM

ARM je rodina procesorových instrukčních sad typu RISC vytvořených společností ARM Holding[6]. Společnost ARM Holding procesory nevyrábí, ale pouze je navrhuje a licencuje ostatním výrobcům, kteří je potom vyrábí. Díky svým přednostem je velmi populární a k firmám, které si návrh procesorů licencují, patří téměř všechny velké společnosti, zabývající se výrobou elektroniky jako například Samsung, Apple a Nvidia. Důvodem proč si tolik firem volí právě ARM je ten, že je velmi jednoduchý, má malou spotřebu energie a díky tomu dominuje trhu s mobilní elektronikou.

Instrukční sady typu RISC se snaží mít všechny instrukce co nejkratší a pokud možno stejně dlouhé, což má za následek to, že se dá jejich vykonávání dobře řetězit. Proto mohou mít ARM procesory jednodušší a efektivnější hardwarovou implementaci, která zabere méně místa na čipu a má menší spotřebu. Jak se výkon ARM procesorů zvyšoval, začalo být reálné jejich nasazení do serverů, kde se v současné době začínají prosazovat.

Do světa superpočítačů zatím nijak výrazně nepronikly, protože do nedávné doby existovaly pouze procesory ARM s 32bitovou instrukční sadou. To se změnilo s příchodem architektury ARMv8, která je již plně 64bitová a může být naplno využita i při výpočtech s dvojitou přesností.

2.2 big.LITTLE

V mobilním odvětví se současně zvyšuje poptávka po vyšším výkonu a zároveň i tlak na delší výdrž na baterii. Jedním z řešení není pouze ladit jednotlivá jádra a snižovat jejich příkon, ale udělat změnu v designu celého čipu a výkonná a úsporná jádra vhodně namíchat. Přesně o toto se snaží heterogenní architektura big.LITTLE od společnosti ARM.

Ta využívá toho, že většinu času není potřeba 100% výkonu procesoru. V čase kdy není velký výkon zapotřebí, jsou používána slabší jádra, která dokáží uspořit dost energie. Pokud

je zapotřebí většího výkonu, jako např. při hrách a vykreslování, jsou zapnuta po krátkou dobu i velká jádra, která mohou škálovat na potřebný výkon.

Tato technologie se může hodit i u clusterů, kdy v časech nečinnosti bude systém fungovat pouze s úspornými jádry a na při spuštění úlohy bude uživateli k dispozici plný výkon.

2.3 Grafické karty

V posledních několika letech je trendem, že surový výpočetní výkon v superpočítačových systémech zastávají nejvíce grafické akcelerátory, jejichž výkon je několikanásobně větší než u procesorů a lze tak pomocí nich vybudovat efektivnější výpočetní systémy. Jejich použití ale není tak jednoduché, protože programování grafických akcelerátorů se liší od programování klasických procesorů.

Projekt k-Wave, zmiňovaný v Úvodu, řeší simulace ultrazvukových vlnění pomocí technologie CUDA na plnohodnotných grafických kartách. Důvod, proč se začala hledat alternativa, je ten, že současná simulační konfigurace má problémy s nasazením v běžném prostředí. Pokud sečteme příkon systému s několika highendovými grafickými kartami, zdroji a Intel Xeon procesory, dostaneme se již k vysokým číslům. Řešením by mohlo být použití velkého počtu zařízení s nízko-příkonovými čipy sestavených dohromady do clusteru. Tyto čipy spojené dohromady by měly mít alespoň srovnatelný výkon jako běžné plnohodnotné karty.

2.4 Anselm a Salomon

Většina výsledků naměřených na postaveném clusteru je porovnávána s čísly získanými přímo v produkčním prostředí na superpočítačových clusterech Anselm a Salomon provozovaných IT4Innovations v Ostravě. V testech byly vždy používány uzly se stejnou hardwarovou konfigurací – bez grafických karet Nvidia či akcelerátorů Intel Xeon Phi.

Anselm Uzly na clusteru Anselm mají dva sockety, ve kterých jsou umístěny osmijádrové procesory Intel Xeon E5-2665 s architekturou Sandy Bridge běžící na 2,4 GHz s maximem v Turbo Boostu na 3,1 GHz. Ve špičce dosahuje jedno jádro maximálního výkonu 19,2 GFLOP/s. Každý uzel má tedy k dispozici 16 jader – technologie Hyper-Threading je vypnutá. Na každý procesor připadá 20 MB L3 cache a 256 kB L2 cache na jádro[11].

Na jednom uzlu je instalováno 64 GB operační paměti, která je rozdělena do dvou NUMA uzlů, kde má každý procesor k dispozici 32GB. Jeden procesor má TDP 115 W. Procesory architektury Sandy Bridge jsou také vybaveny SIMD vektorovou jednotkou AVX, která má šířku 256 bitů a umí pracovat s čísly jak v jednoduché tak i dvojité přesnosti.

Salomon Uzly na clusteru Salomon mají také dva sockety, ve kterých jsou umístěny dvanáctijádrové procesory Intel Xeon E5-2680v3 běžící na frekvenci 2,5 GHz s maximem v Turbo Boostu na 3,3 GHz. Každý uzel má tedy k dispozici 24 jader s vypnutou technologií Hyper-Threading. Na každý procesor připadá 30 MB L3 cache[12].

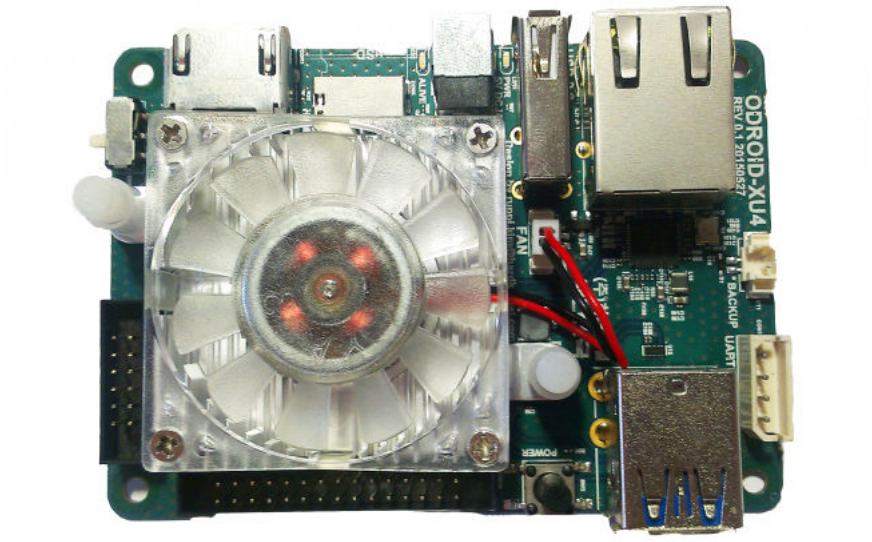
Na jednom uzlu je instalováno 128 GB operační paměti, která je rozdělena do dvou NUMA uzlů, kde každý procesor má k dispozici 64 GB. Jeden procesor má TDP 120 W. Tyto už mají podporu pro AVX2 – vylepšení původního AVX.

Kapitola 3

Použitý hardware

V této části jsou uvedeny a popsány všechny důležité parametry kitu Odroid XU4, který je osazen procesory ARM. Většina specifikací byla získána z datasheetů a technických manuálů společností ARM a Hardkernel.

Na kitu je použit procesor Samsung Exynos 5422, který lze najít např. i v telefonech Samsung Galaxy S5. Čipy Samsung Exynos jsou licencované od společnosti ARM a jsou založené na architektuře big.LITTLE s jádry Cortex A15 v silnějším clusteru označovaném *big* a jádry Cortexem A7 v úsporném clusteru označovaném *LITTLE*. Tento procesor byl uveden na trh již v roce 2014, architektura jader Cortex A7 a A15 - ARMv7-a je ale ještě o několik let starší. Bohužel společnost Samsung je velmi skoupá na uveřejňování technických detailů o svých procesorech a většinou nejsou k dispozici podrobné specifikace.



Obrázek 3.1: Kit Hardkernel Odroid XU4[10].

Kit nabízí dobré možnosti připojení. Obsahuje gigabitový Ethernet, což je důležité pro rychlé předávání dat po síti mezi uzly. Osazeno je i rozhraní USB3.0, díky kterému je možné připojit další periferie jako rychlý externí disk, který lze poté použít jako síťové úložiště.

Tabulka 3.1: Specifikace kitu Odroid XU4 [10].

	Odroid XU4
Ethernet [Mb/s]	1000
paměť LPDDR3 [MB]	2048
frekvence paměti [MHz]	825
propustnost paměti [GB/s]	13.2

Tabulka 3.2: Specifikace procesorové části Odroid XU4 [16].

Exynos 5422	A15 cluster (big)	A7 cluster (LITTLE)
počet jader	4	4
revize procesoru	r3p2	r0p3
výrobní technologie	28nm HKMG	28nm HKMG
L3 cache	není	není
L2 společná cache [kB]	2048	512
L1 cache - data [kB]	32	32
L1 cache - instrukce [kB]	32	32
vfpv4 + NEON	ano	ano
max. frekvence [MHz]	2000	1400
min. frekvence [MHz]	200	200

Tabulka 3.3: Specifikace grafické části Odroid XU4 [4].

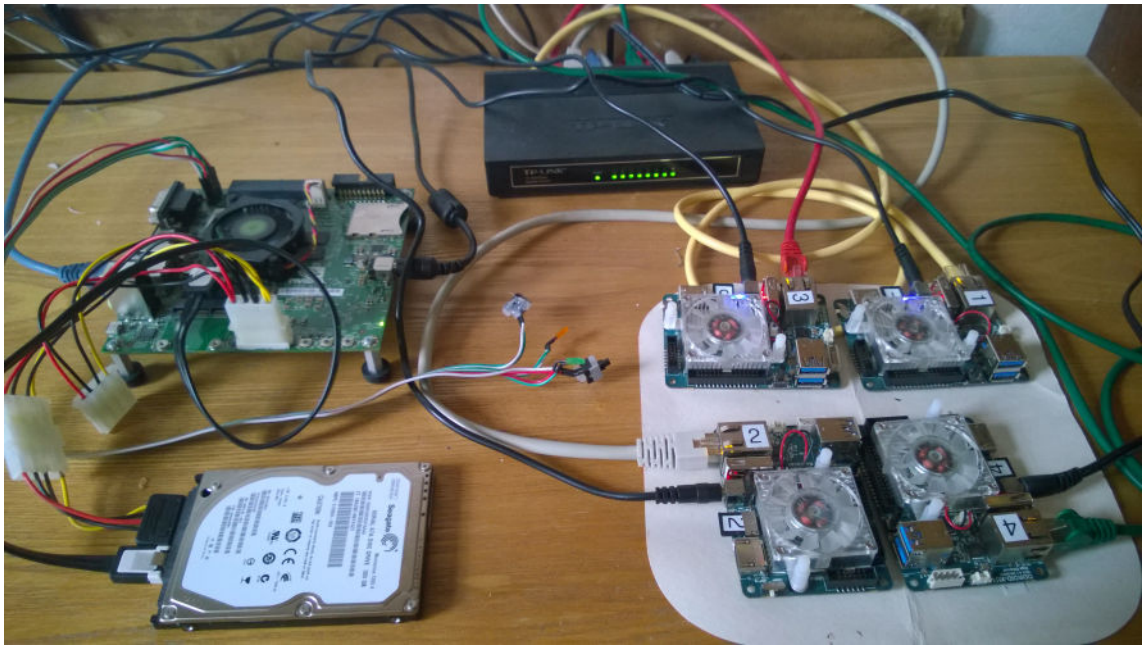
	ARM Mali^{TM}-T628 MP6
architektura	Midgard 2nd gen.
počet jader	6
max. frekvence [MHz]	600
výkonnost [GFLOP/s]	325
OpenCL	1.1

Kapitola 4

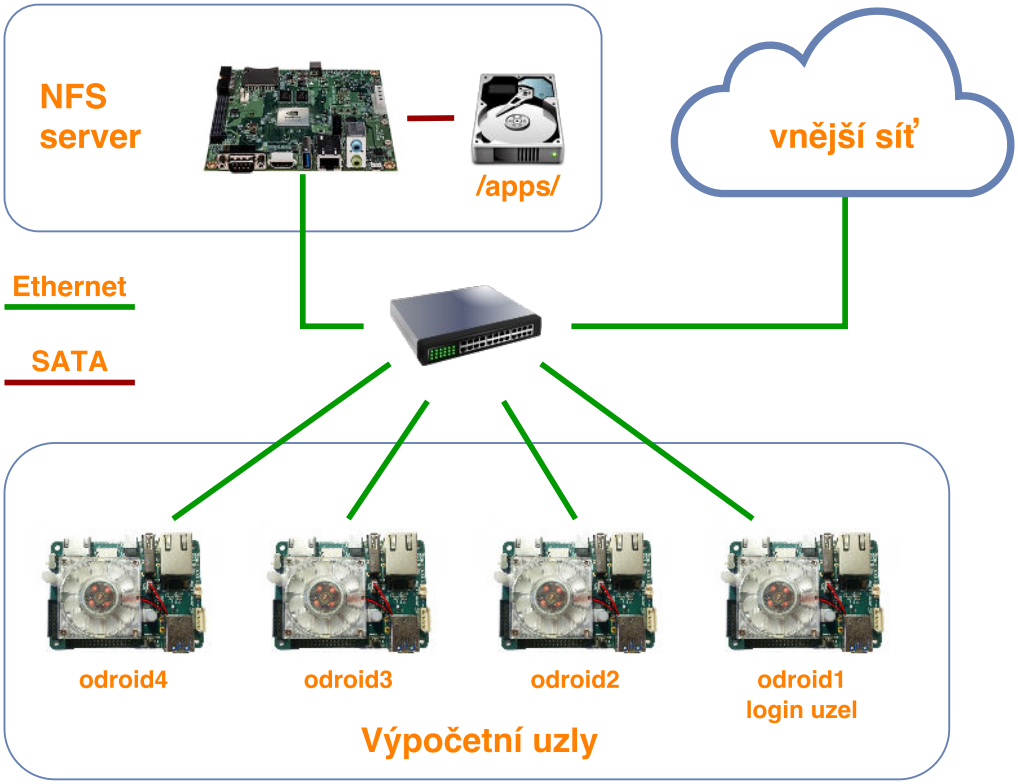
Návrh clusteru

V této kapitole je rozebráno, jak bylo postupováno při instalaci hardwarového a softwarového vybavení clusteru. Pokud to není uvedeno přímo v práci, tak detailní postupy všech nastavení a příkazů jsou popsány na wiki stránkách v repositáři s prací. Stažená kopie repozitáře je dostupná na přibaleném CD.

Cluster obsahuje celkem 4 výpočetní uzly - kity Hardkernel Odroid XU4, z nichž jeden je vždy vybrán jako přihlašovací (tzv. login uzel). Na ten se přihlásí uživatel pomocí SSH nebo VNC a z něj spouští úlohy. Na ostatní uzly se lze také manuálně přihlásit, ale nelze z nich vydávat úlohy pro plánovač. Všechny 4 uzly uživateli jsou k dispozici pro výpočty. Dále je ke clusteru připojen další uzel, který slouží jako síťové úložiště s aplikačním vybavením.



Obrázek 4.1: Fotografie skutečného zapojení Odroid XU4 clusteru.



Obrázek 4.2: Schéma zapojení clusteru.

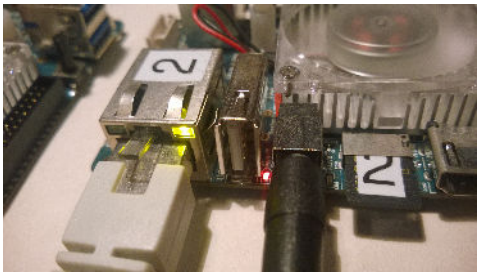
4.1 Síťová infrastruktura

Jednotlivé uzly jsou propojeny dohromady pomocí sítě Ethernet, který se na kitech nachází v gigabitové verzi. Pro účely tohoto clusteru je to dostatečná rychlost. K propojení slouží 8portový gigabitový switch, kterým jsou spojeny všechny uzly dohromady. Zapojení je provedeno podle schématu na Obrázku 4.2. Síťové datové úložiště je fyzicky připojeno k datovému serveru a je pomocí síťové protokolu NFS k dispozici všem výpočetním uzlům. Na něm se nacházejí některá uživatelská data a nainstalované sdílené programové vybavení.

Síťová karta na Odroidu je bohužel připojena pouze pomocí sběrnice USB, takže její provoz je znatelně zatížen latencí. Aby byla tato nevýhoda překryta, musí být posílána data po větších částech, ve kterých se latence ztratí.

Datové úložiště Jako datové úložiště slouží kit Nvidia Jetson TK1, ze kterých měl být původně celý cluster postaven. Jeho procesor je podobný tomu v Odroidu, je také osazen čtyřmi ARM Cortexy A15 a jedním dalším úsporným jádrem. Navíc má v sobě grafickou část se 192 CUDA jádry architektury Kepler. Jako datový uzel byl zvolen proto, že má přímo vyvedený SATA konektor a k němu připojené disky tak mají dobrý výkon. Díky úspornému jádru má v případě nečinnosti malou spotřebu a nezvedá příliš celkový příkon clusteru. Jako úložiště pro data byl použit plotnový disk Seagate ST9320423AS se 7200 rpm a kapacitou 320 GB, který poskytuje dostatečný výkon a kapacitu.

Jako velmi praktická záležitost se ukázalo označení dvojic kitů a příslušných microSD karet pomocí štítků, jak je to vidět na Obrázku 4.3. Nedochází tak k nechtěným záměnám kitů a následným problémům s MAC adresami.



Obrázek 4.3: Fotografie štítkování kitů Odroid XU4.

4.2 Klíče pro komunikaci

Při použití prostředků pro paralelizaci výpočtů, jako např. MPI a u plánovače úloh pro cluster je důležité, aby si mohly uzly mezi sebou automatizovaně vyměňovat data, nejlépe pomocí SSH. Samozřejmě každý uzel může mít svoje unikátní klíče, avšak správa systémů by byla při větším počtu uzlů velmi náročná - každý uzel by musel mít klíče všech ostatních uzlů ve svém seznamu autorizovaných klíčů `.ssh/authorized_keys`. Aby bylo přidávání a odebírání uzlů z clusteru co nejjednodušší, používají všechny uzly jeden stejný pár privátního a veřejného klíče a zároveň mají veřejný klíč z tohoto páru všechny uzly uložený jako autorizovaný. Tento model je použit i u velkých superpočítačových clusterů. Tím je také umožněno to, aby všechny uzly používaly ten samý obraz operačního systému.

Všechny kity tedy používají stejný obraz operačního systému. Jediné nastavení, kterým se kity mezi sebou liší, je síťový identifikátor `hostname`, který je vždy potřeba změnit. U klientských uzlů se kromě toho již jen vypnou serverové služby. Tento model je výhodný zejména pro správu clusteru a případné přidávání či výměnu uzlů.

Soubor 4.1: `/etc/hosts`

```
127.0.0.1       localhost
127.0.0.1       odroid1
::1             localhost ip6-localhost ip6-loopback
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters
```

4.3 Operační systém

Společnost Hardkernel poskytuje pro svůj kit předinstalované binární obrazy se systémem Ubuntu, v době psaní založený na verzi 15.10. Ten přichází předpřipravený se všemi potřebnými ovladači. Na kit lze nahrát i operační systém Android, který však není pro clustery vhodný.

Jedním z cílů bylo i snadné řešení záloh systému. To je velmi snadné, protože kit využívá jako místní úložiště microSD karet, jejichž obsah lze velmi jednoduše zálohovat a přehrávat.

4.4 Plánovač

Na velkých clusterech je potřeba rozdělit procesorový čas mezi několik uživatelů. Ti mohou spouštět svoje úlohy ve formě předem připravených scriptů, které spustí a jsou předány plánovači, který je zařadí do fronty úloh. Od té chvíle se již uživatel nemusí o svoji úlohu starat a až na ni dojde řada, je spuštěna a uživatel může být informován o jejím dokončení.

Jednotlivé plánovače jsou si většinou velmi podobné v tom, co nabízejí, ale v použití některých příkazů se drobně liší. Cílem bylo, aby byl navržený systém pokud možno co nejvíce kompatibilní s již existujícími scripty, které jsou používány na superpočítačích Anselm a Salomon.

Na nich běží plánovač PBS Professional[1], který je však nutné mít licencovaný a proto byl jako plánovač zvolen balík Torque[2], který je stejně jako PBS Professional pokračovatelem původního projektu PBS a jsou od něj dostupné zdrojové kódy, takže jej není problém přeložit na architektuře ARM.

V době psaní práce se objevila informace, že PBS Professional by měl být vydán i jako Open-source a to přibližně v polovině roku 2016 [1], což je již po termínu odevzdání této práce. Při budoucím budování podobného clusteru bych zvážil při výběru i tuto možnost.

Torque v sobě v základu obsahuje jednoduchý plánovač, který plánuje systémem na bázi FIFO, tedy skládá jednotlivé úlohy do fronty a ve stejném pořadí se dostávají řadu. Jde však nahradit balíkem MOAB, který již umí řadit efektivněji podle zadaných priorit a při reálném nasazení na clusteru s více uživateli by byl MOAB doporučovanou variantou.

Přestože obsahuje každý uzel celkově 8 jader, má Torque u každého uzlu k dispozici pouze 4 jádra, jak je vidět v konfiguraci v Souboru 4.2. Záměrně ale není definováno, která jádra to jsou, a je necháno na uživateli, jak s nimi bude pracovat pomocí prostředků pro distribuované výpočty, jako např. MPI.

Soubor 4.2: `/var/spool/torque/server_priv/nodes`

```
odroid1 np=4
odroid2 np=4
odroid3 np=4
odroid4 np=4
```

4.5 Prostředky pro správu softwaru

Jedním z úkolů pro administrátory HPC clusteru je i instalace softwaru pro jeho uživatele. Vzhledem k tomu, že superpočítačové systémy jsou zaměřeny na optimalizaci na co nejvyšší výkon a rychlost, snaží se, jak to jde, využívat nejnovější verze kompilátorů, knihoven a softwaru s ručně vyladěnými konfiguracemi speciálně pro architekturu daného clusteru. Typicky je proto preferována možnost si software samostatně kompilovat na strojích, na kterých bude použit, než používat generické binární balíčky z repositářů.

Protože cluster většinou využívá větší množství uživatelů, často s různými, mnohdy protichůdnými potřebami, tak není dostačující udržovat jednu verzi softwarového balíku a pouze ji upgradovat na vyšší verze. Přítomno je často několik verzí, přeložených různými kompilátory s různými konfiguračními parametry a s podporou pro různé externí knihovny.

[1] http://www.pbsworks.com/

[2] http://www.adaptivecomputing.com/products/open-source/torque/

Příkladem může být několik implementací MPI standardů - knihovny OpenMPI vs. MVAPICH2. Tradiční balíkovací programy obsažené v distribucích však pro tento účel nejsou příliš vhodné.

Jednoduchým a účinným řešením problému jsou *moduly prostředí*, v originále environment modules. Ty umožňují jednoduché zprovoznění programů a přepínání mezi nimi jen změnami v proměnných prostředí (environment variables) jako jsou `$PATH` a mnohé další, které mohou být nastaveny speciálně pro jednotlivé programy (např. `$CC` pro nastavení cesty k gcc). Jedním z nástrojů, které moduly využívají je softwarový balík EasyBuild, který byl použit i při řešení této práce.

4.5.1 EasyBuild

EasyBuild je framework napsaný v jazyce Python, který slouží k překladu a instalaci hlavně vědeckého softwaru pro HPC systémy. Zároveň nainstalovaný software spravuje a připravuje jej k užití pomocí modulů prostředí, které jsou detailněji popsány v Sekci 4.5.2. Samotný EasyBuild je také nainstalován jako modul.

Jeho výhodou je, že instalace programů probíhá zcela autonomně a celý proces je téměř nezávislý na uživateli. To znamená že EasyBuild je schopen si sám stáhnout zdrojové kódy pro danou verzi programu, pokud je potřeba tak aplikovat patche, načíst zvolený kompilátor, sám provést konfiguraci, překlad a instalaci. Součástí je samozřejmě i testování a vytvoření modulu. A to celé je schopen rekurzivně opakovat tehdy, pokud má instalovaný software závislosti na dalších programech.

Easyconfig Informace o všem potřebném k instalaci jednoho programu jsou uloženy v tzv. *easyconfig* souboru s příponou `.eb`. Jejich databáze je dodávána přímo s EasyBuildem a jsou s každou verzí pravidelně aktualizovány, aby reflektovaly novinky v programech. Ukázku je možné vidět v Souboru 4.3. Jméno každého easyconfig souboru obsahuje čtyři údaje, které jej charakterizují. U ukázkového souboru `HDF5-1.8.16-foss-odroid2016a.eb` to jsou:

- `HDF5` - název instalovaného softwaru
- `1.8.16` - verze instalovaného softwaru
- `foss` - název toolchainu použitého pro překlad
- `odroid2016a` - verze toolchainu použitého pro překlad

Základní použití je velmi jednoduché. Využívá příkaz `eb` a jméno easyconfigu. Parametr `-robot` zapíná automatické zjišťování závislostí - jinak EasyBuild očekává, že všechny programy, na kterých existuje závislost, jsou již přítomny. Pokud chybí i toolchain, EasyBuild se jej pokusí doinstalovat.

```
eb HDF5-1.8.16-foss-odroid2016a.eb -robot
```

Znovupoužitelnost Síla EasyBuildu spočívá mimo jiné i ve znovupoužitelnosti easyconfig souborů. Pokud v repositáři existuje easyconfig pro program ve verzi 1.8.16, je jednoduše možné jej využít k instalaci verze 1.18.17 pouhým přepínačem `-try-software-version=1.8.17`. Takto probíhá i update samotného EasyBuildu na novější verzi. To samé platí i o použití jiného toolchainu, který lze změnit přepínačem `-try-toolchain=GCC,5.3.0`. Toto nastavení je samozřejmě aplikováno i na všechny závislosti, takže není problém takto překompilovat celý strom závislostí jediným příkazem.

Architektura ARM Bohužel hlavní zaměření EasyBuildu je na architekturu x86-64, tedy hlavně na procesory Intel a v posledních verzích i na platformu Cray. Mnoho easyconfig souborů proto není použitelných a je potřeba je před použitím nejprve upravit. Jedná se hlavně o přepínače pro překladače neslučitelné s fungováním na 32bitové platformě ARMv7-a, např. `-m64`, přímé použití adresářů `lib64` či vynechání platformy ARM při větvení podle platforem. Výběr 64bitového kitu by tedy výrazně zjednodušil práci a zvýšil kompatibilitu.

Přínosem této práce je tedy také sada upravených easyconfig souborů a návodů, které obsahují změny potřebné pro fungování na architektuře ARMv7-a.

Soubor 4.3: Ukázkový easyconfig soubor `HDF5-1.8.16-foss-odroid2016a.eb`

```
name = 'HDF5'
version = '1.8.16'
homepage = 'http://www.hdfgroup.org/HDF5/'
description = """HDF5 is a unique technology suite that makes... """
toolchain = {'name': 'foss', 'version': 'odroid2016a'}
source_urls = ['www.hdfgroup.org/ftp/HDF5/releases/hdf5-%(version)s/src']
sources = [SOURCELOWER_TAR_GZ]
patches = [
    'HDF5-1.8.15_configure_intel.patch',
    'configure_libtool.patch',
]
buildopts = 'CXXFLAGS="$CXXFLAGS -DMPICH_IGNORE_CXX_SEEK"'
dependencies = [
    ('zlib', '1.2.8'),
    ('Szip', '2.1'),
]
moduleclass = 'data'
```

Easyblocks Pokud instalace nějakého softwaru vyžaduje složitější přístup, tak přichází na řadu funkce, která má název *easyblocks*. Každý easyconfig může obsahovat jméno easyblocku, který bude použit pro instalaci. Ten říká, jak bude instalace probíhat a popř. přidávat či ubírat některé kroky. Pokud není žádný přítomen, použije se standardní easyblock `ConfigureMake`, který zvolí běžný postup instalace `configure - make - make install`.

4.5.2 Moduly prostředí

Pokud není software instalován do standardních systémových adresářů, tak je pro jeho zařazení do systému často modifikována proměnná prostředí `$PATH` tak, aby obsahovala lokace binárních souborů. Také mohou být modifikovány i jiné proměnné jako `$LD_LIBRARY_PATH` pro knihovny pro run-time či `$CPATH` pro include adresáře. Naivní přístup by mohl vypadat tak, že pro každý software bude existovat script, který tyto proměnné nastaví.

Moduly prostředí tuto myšlenku rozšiřují a zavádějí jednotnou formu těchto scriptů zvané *modulefiles* a příkazů, které s nimi pracují, hlavně příkaz `module`.

Implementace K dispozici jsou dva programy, které zajišťují funkci modulů prostředí – `environment modules` a `lmod`. Zatímco `lmod` nabízí větší možnosti pro vytvoření hierarchie a tříd modulů a probíhá u něj stálý vývoj, tak `modules` nabízejí jen fixní dělení podle

adresářů. Zato jsou již několik let na stejné verzi a jsou prověřeny dobou. Zvolena byla varianta `modules`, protože pro tento projekt stačí základní požadavky na hierarchii. Každý modul tak spadá do třídy, která je specifikována v easyconfigu v nastavení `moduleclass`, jak je vidět v Souboru 4.3.

Základní příkazy Základním příkazem pro použití je příkaz `module load`, který načte modulefile pro zvolený software a nastaví cesty tak, aby byl software připraven k použití.

```
module load OpenMPI
```

Velkou výhodou je i to, že operace s proměnnými prostředí jsou většinou nedestruktivní, takže k příkazu `module load` existuje i opačný příkaz `module unload`, který naopak všechny cesty vrátí do původního stavu.

K výpisu a vyhledání existujících modulů slouží příkaz `module avail`. Řetězec za lomítkem značí číslo verze a toolchain, kterým byl software kompilován, popř. další software, který byl ke kompilaci použit.

```
module avail OpenMPI
OpenMPI/1.10.2-GCC-5.3.0
OpenMPI/1.10.2-intel-2015b
```

4.5.3 Toolchain

Některé programy vyžadují ke své kompilaci kromě překladače i několik dalších externích knihoven a programů. Aby byl proces jejich instalace používání jednodušší, byly vytvořeny tzv. *toolchainy* (compiler toolchain). Toolchain je zastřešením sady programů a knihoven potřebných pro překlad a provozování jiných aplikací. Základem každého toolchainu je překladač, tedy GNU/GCC, Clang nebo překladač od společnosti Intel. Dále je většinou obsažena MPI knihovna a popř. i další knihovny – pro BLAS nebo podporu CUDA. Každý toolchain má svoji zkratku, která je používána u jmen easyconfigů a modulefiles.

Intel toolchain bohužel není dostupný na architektuře ARM. Pro kity byly nainstalovány tyto toolchainy:

- `GCC - GCC`
- `gompi - GCC, OpenMPI`
- `foss - GCC, OpenMPI, FFTW, OpenBLAS, ScaLAPACK`

Pro EasyBuild není problém vytvářet vlastní toolchainy. Pro demonstraci byla vytvořena další verze toolchainu `foss`, verze `odroid2016a`, obsahující jako základ GCC 5.3.0 a OpenMPI 1.10.2, se kterým byly překládány všechny další programy.

Překladač GNU/GCC

Různé projekty mohou vyžadovat jako závislost odlišné verze překladačů pro svoje správné fungování. Nejde ale jen o číslo verze, ale i o podporované funkce. Např. systémový překladač na kitu, GNU/GCC 5.2.1, není přeložen s podporou pro Fortran, což může být pro některý vědecký software problém. Dále může jít např. o zapnutí podpory různých optimalizací jako LTO - link time optimizations či podporu pro ld.gold linker, které mohou být v generických balících či předkompilovaných toolchainech vypnuté. Zvládnutí překladu překladače je tedy celkem klíčová záležitost.

Protože úprava easyconfig a easyblock souborů GNU/GCC pro architekturu ARMv7-a by byla příliš složitá, bylo GNU/GCC přeloženo zvlášť a zprovozněno jako *externí modul*. Ty nemusejí být instalovány přes EasyBuild, pouze je potřeba je nainstalovat do adresářů s ostatními programy a je potřeba pro ně manuálně vytvořit moduly.

Povedlo se přeložit GNU/GCC ve verzi 5.3.0, pomocí kterého pak byla překládána veškerá další softwarová výbava kitu. Parametry použité ke konfiguraci jsou v Souboru 4.4.

Soubor 4.4: Konfigurační parametry překladače GNU/GCC 5.3.0 získané příkazem gcc -v

```
Target: armv7l-unknown-linux-gnueabihf
Configured with: ../gcc-5.3.0/configure --prefix=/apps/software/GCC/5.3.0
--enable-languages=c,c++,fortran --enable-multilib --enable-lto
--enable-shared=yes --enable-static=yes --enable-threads=posix
--with-arch=armv7-a --with-float=hard --with-fpu=vfpv4 --with-mode=thumb
--enable-plugins --with-plugin-ld=ld.gold --enable-gold=default
--enable-ld --with-arch-directory=arm --with-system-zlib
--enable-checking=release --disable-libquadmath --disable-sjlj-exceptions
--disable-werror --disable-libitm --enable-linker-build-id
--enable-clocale=gnu --enable-libstdcxx-time=yes --enable-libstdcxx-debug
--with-default-libstdcxx-abi=new --enable-gnu-unique-object

Thread model: posix
gcc version 5.3.0 (GCC)
```

Zajímavé by také mohlo být porovnání s referenčním překladačem od společnosti ARM – ARM Compiler [3]. Ten by mohl mít zajímavý potenciál při optimalizaci, obzvlášť v oblasti generování vektorových instrukcí NEON, u kterých by mohl být potenciálně lepší, než GNU/GCC. Tento překladač lze používat pouze na platformě x86 nebo x86-64 jako cross-compiler, nikoli na kitu samotném. Bohužel je také problém s jeho licencováním a publikací jeho výsledků.

4.6 Message passing interface

Pro MPI byla použita knihovna OpenMPI ve verzi 1.10.2 a s ní byly vytvořeny i přidružené toolchainy pro EasyBuild. Pro instalaci je dobře možné použít připravené easyconfig soubory, ovšem je potřeba zvolit tzv. *no-OFED* verzi s vypnutou podporou sítě InfiniBand. Easyconfig s OpenMPI má závislost na balíku hwloc, jehož easyconfig je potřeba upravit a přidat řádek `configopts = "-disable-libnuma"` – viz problém níže.

U MPI se projevil jeden neřešitelný problém a to absence podpory NUMA v systému. Názvem NUMA, neboli "Non-uniform memory access", se označují hlavně víceprocesorové systémy, které v sobě obsahují více pamětí, ke kterým má procesor(y) rozdílné přístupové časy. Na kitu tato architektura není, kit má jen jednu hlavní paměť. MPI ale pro některé své funkce vyžaduje, aby v jádru systému byla podpora NUMA. Bez podpory NUMA v jádru nelze používat následující parametry scriptu `mpirun` a `mpiexec`:

- `-bind-to`

[3] http://ds.arm.com/downloads/compilers/

- `-cpu-set`

Toto je problém např. tehdy, pokud je potřeba MPI procesy spouštět pouze na výkonnějších Cortex A15 jádrech a slabší jádra nevyužívat. Nehledě na to, že díky tomuto problému mohou různé projekty přestat fungovat.

Bohužel v kódu jádra pro Odroid kity se podpora NUMA v současné době nevyskytuje a platí to pro všechny systémy postavené 32bitové architektuře ARMv7-a. Pro rozumné použití MPI při stavbě potenciálního budoucího clusteru bych doporučil zvolit architekturu AArch64, která by již měla mít podporu pro NUMA implementovanou.

Nejjednodušším řešením by bylo úsporná jádra vypnout pomocí vložení hodnoty `0` do systémového souboru `/sys/devices/system/cpu/cpuN/online`. Toto je možné provést pro všechna jádra kromě jádra `cpu0`, tedy první úsporného jádra Cortex A7.

Dalším řešením je pro přiřazování MPI procesů k jádrům používat tzv. *rankfiles* a parametru `-rf rankfile`. Ukázkový rankfile je vidět v Souboru 4.5. Rankfile přesně specifikuje každému MPI ranku hostname uzlu a číslo nebo rozsah jader, na kterých může být spuštěn. Slot udává, který z procesorových clusterů se má použít, buďto Cortex A7 (`slot=0`) nebo Cortex A15 (`slot=1`). Toto řešení bylo nakonec zvoleno při testování.

Soubor 4.5: Ukázkový rankfile, přiřazující každému MPI ranku hostname uzlu a číslo nebo rozsah jader, na kterých může být spuštěn.

```
rank 0=odroid1 slot=1:0
rank 1=odroid1 slot=1:1
rank 2=odroid1 slot=1:2
rank 3=odroid1 slot=1:3
rank 4=odroid2 slot=1:0-3
rank 5=odroid2 slot=1:0-3
rank 6=odroid2 slot=1:0-3
rank 7=odroid2 slot=1:0-3
```

Kapitola 5

Testování výkonu

Testování bylo zaměřeno na několik oblastí, ve kterých bylo potřeba zjistit jak si kit Odroid XU4 vede. Jedná se jednak o změření parametrů jednoho kitu a následné srovnání výsledků, zejména s jeho konkurencí. Dále byl také změřen výkon všech čtyř kitů propojených sítí do clusteru a byly porovnány jeho vlastnosti s velkými clustery jako jsou Anselm a Salomon.

5.1 Přesnost měření

Přesnost měření bývá negativně zatížena několika faktory. V této časti je uvedeno, jaká opatření byla učiněna, aby měření byla prováděna co nejpřesněji.

Škálování procesoru

Procesory po většinu času běhu nekonají žádnou užitečnou činnost a proto je záhodno, aby při nízké poptávce po výkonu ze strany uživatele i systému byla frekvence procesoru cíleně snížena, a procesor tak měl menší příkon. To se projeví příznivě na spotřebě a také na cenách za spotřebovanou energii a vyzářeném teplu, které není potřeba odvádět. A nejvíce také na výdrži na baterii u mobilních zařízení.

Je však obtížné predikovat, kdy se má frekvence procesoru snížit, protože není dopředu známá poptávka po výkonu. Proto je frekvence snižována po menších skocích po určitých časových krocích, aby při drobném snížení poptávky po výpočetním výkonu procesor hned nezpomalil na nejmenší možnou úroveň, ale snižoval frekvenci postupně, a nebyl problém se zasekáváním. Tato technologie je označována jako CPU Throttling, česky také dynamické škálování frekvence. K tomu jak rychle a po jakých krocích se bude frekvence měnit, existuje několik odlišných strategií a v operačním systému GNU/Linux jsou tyto jednotlivé strategie nazývány jako *governory*.

U benchmarků je ale cílem změřit čistý výkon a nemít výsledky zatíženy neustálým měněním frekvence procesoru. Nejvíce se toto nepříjemně projeví na začátcích testů, kdy jde procesor z klidové hodnoty na maximální zatížení a do doby než se frekvence zvedne na maximální hodnotu, mohou být naměřené hodnoty nepřesné a konečné výsledky zavádějící.

Proto byl po celou dobu testování v systému nastaven governor *performance*, který zajišťuje, že procesor neškáluje a běží stále na nastavené maximální frekvenci, většinou té nejvyšší možné.

Teoretický výkonový strop

Při testování výkonu je dobré vědět, nakolik se naměřené výsledky přibližují k teoretickému výkonovému stropu, kterého je kit schopen dosáhnout. Uveden je zde pouze pro výkonnější jádra Cortex A15. Instrukční pipeline Cortexu A15 umí v jednom taktu získat z L1 instrukční cache až 3 instrukce a vydat instrukce až z 8 různých front. Dvě z těchto front jsou určeny pro vydání floating-point instrukcí nebo instrukcí ARM NEON a vydání jsou schopny provést každý takt[19]. ARM NEON je 128bitová SIMD vektorová jednotka, která je schopna provést až 4 single-precision floating-point operace zároveň. To je dohromady maximálně 8 vydaných single-precision floating-point operací za takt.

Rovnice 5.1 demonstruje teoretický výkonový strop pro floating-point výpočty v single-precision. K němu je ale možné se jen přiblížit a to pouze dokonalým instrukčním mixem s minimem load/store operací. Rovnice 5.2 ukazuje totéž pro double-precision.

$$4\ \text{jádra} \cdot 2\ \text{GHz} \cdot \frac{2\ \text{vydané instrukce}}{\text{takt}} \cdot \frac{4\ \text{SP operace}}{\text{NEON instrukci}} = 64\ \text{GFLOP/s} \tag{5.1}$$

$$4\ \text{jádra} \cdot 2\ \text{GHz} \cdot \frac{2\ \text{vydané instrukce}}{\text{takt}} \cdot \frac{1\ \text{DP operace}}{\text{FPU instrukci}} = 16\ \text{GFLOP/s} \tag{5.2}$$

5.2 Linpack

Linpack je benchmark, který dokáže změřit výpočetní výkon jednotky pro výpočty v plovoucí řádové čárce (neboli floating-point jednotky) jednoho jádra[15]. Benchmark měří, za jak dlouhou dobu dokáže jednotka spočítat soustavu lineárních rovnic $Ax = b$ a z tohoto čísla spočte hodnotu GFLOP/s.

Nevýhoda tohoto benchmarku je, že jeho výsledky dosahují přibližně poloviního výkonu oproti benchmarku HPL, protože nevyužívá optimalizovaných rutin pro BLAS a vše si překládá sám bez náročných optimalizací. Z této nevýhody se však stává výhoda pro následující test, který se snaží analyzovat různé kombinace přepínačů pro překladače, protože se může naplno projevit jejich efekt.

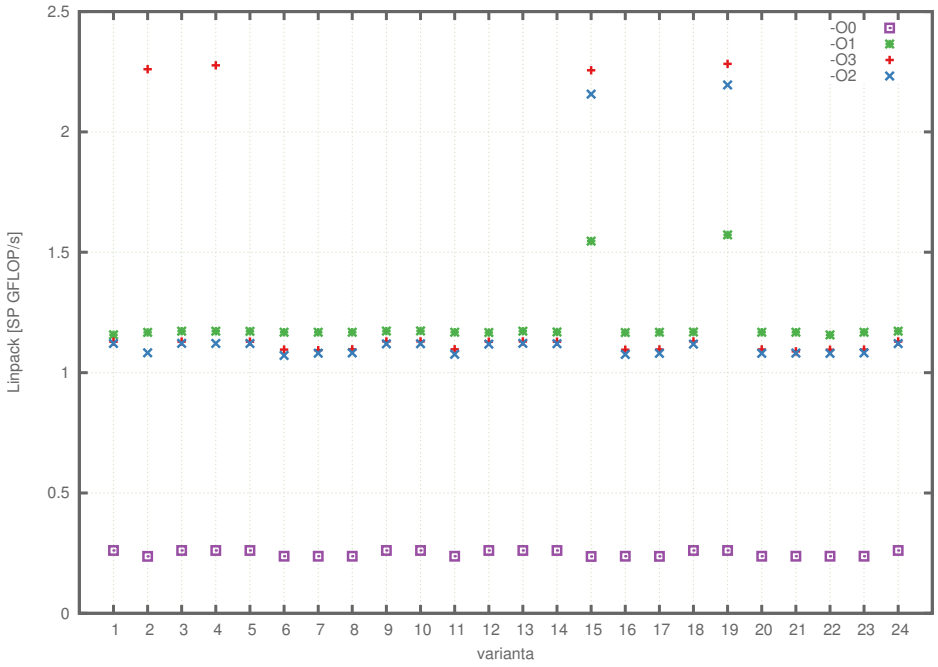
Test přepínačů pro překladač

Aby bylo možné změřit nejlepší hodnotu GFLOP/s, které je procesor schopen, je potřeba znát nejlepší kombinaci parametrů pro kompilátor pro architekturu kitu. Použitý kompilátor GNU/GCC podporuje celou řadu kompilačních parametrů, které mu mohou pomoci při optimalizaci. Jedná se o parametry povolující různé optimalizace a určující jak se bude pracovat s jednotkou plovoucí řádové čárky, přičemž GNU/GCC již defaultně používá parametry `-mfloat-abi=hard` a `-march=armv7-a`. Parametry, jejichž kombinace byly zkoušeny, jsou následující:

- `-O0, -O1, -O2, -O3`
- `-mtune=cortex-a15`
- `-mfpu={vfpv3, vfpv4, neon-vfpv4}`
- `-ffast-math`

- `-ftree-vectorize`

Původním záměrem bylo do testu zahrnout i testy různých ABI - application binary interfaces floating-point jednotky `-mfloat-abi={soft, softfp, hard}`, protože u testů kitu Nvidia Jetson TK1 (také s Cortexy A15), vykazovaly výsledky při použití těchto nastavení zajímavé rozdíly. Systém na kitu však podporuje pouze `-mfloat-abi=hard`, tedy předávání parametrů funkcí přes FPU registry.



Obrázek 5.1: Výsledky benchmarku Linpack v jednoduché přesnosti pro různé parametry překladače na Odroidu XU4. Přesné popisy variant jsou v Příloze A.

Výsledky

V testu byly vyzkoušeny všechny kombinace parametrů, včetně variant bez nich. Na Obrázku 5.1 jsou výsledky pro Linpack v single-precision, tedy s desetinnými čísly typu `float`. Velikost problému byla použita na základě výsledků následujícího testu 900. Každá z funkcí reprezentuje jednu ze základních úrovní optimalizace `-O0, -O1, -O2, -O3`. Na ose x jsou čísla jednotlivých testů. Hned z počátku je vidět, že nemá smysl spouštět kompilátor bez optimalizací (varianta `-O0`), bez nichž se žádný výsledek nedostal výš, než 0,25 GFLOP/s. Výsledky, jež byly kompilovány různými kombinacemi přepínačů `-mtune=cortex-a15, -mfpu={vfpv4, vfpv3},` se umístily mezi 1–1,2 GFLOP/s. Jejich použití tedy nemá velkého významu, protože velmi podobně se umístily i varianty bez nich. Přepínač `-ffast-math` v kombinaci s nimi většinou znamenal nepatrný posun výkonu směrem nahoru.

Zajímavým výsledkem je, že použití přepínačů `-O2, -O3` oproti `-O1` nemá většího významu a výsledný výkon je velmi podobný a jejich použití nebude v podobných aplikacích, kde je většina operací prováděna nad čísly s plovoucí řádovou čárkou, vidět.

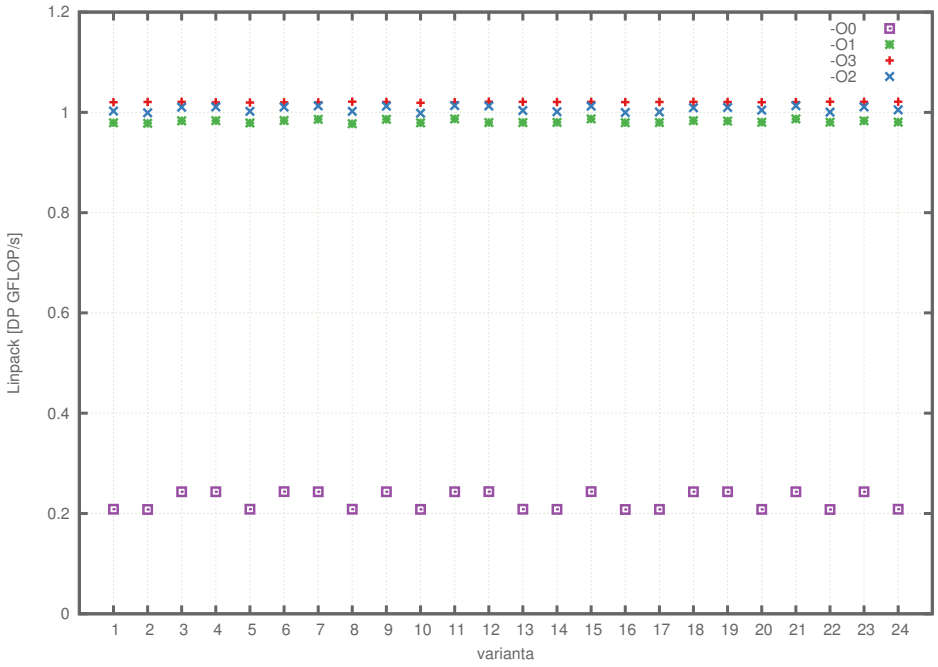
Nejlepších výsledků dosahuje Linpack při použití SIMD vektorizační jednotky ARM NEON[5]. Jedná se o všechny výsledky nad hranicí 1,5 GFLOP/s. V testech se ukázalo, že

základní kombinace, při které se lze dostat na vysoké výsledky jsou tyto:

- `-O3, -mfpu=neon-vfpv4, -ffast-math`
- `-O2, -mfpu=neon-vfpv4, -ffast-math, -ftree-vectorize`
- `-O1, -mfpu=neon-vfpv4, -ffast-math, -ftree-vectorize`

Přepínač `-O3` v sobě již zahrnuje volbu `-ftree-vectorize`, která vektorizaci zapíná. Přepínač `-ffast-math` konfiguruje kompilátor tak, že není nucen dodržovat normu IEEE 754 pro počítání s čísly s plovoucí řádovou čárkou a tím může být výsledný program potenciálně urychlen, protože se např. nemusí dodržovat asociativita operací. Je ale potřeba hlídat přesnost výsledků v použitých programech, protože výsledky se mohou lišit od programů přeložených se striktním vynucením normy IEEE 754! V dalším přepínači `-Ofast` je použití `-ffast-math` již zahrnuto, takže jej u něj není potřeba psát explicitně znovu.

Nejlepším kombinací přepínačů se stala kombinace `-O3, -mfpu=neon-vfpv4, -ffast-math` s výsledkem 2,28 GFLOP/s s náskokem téměř 0,1 GFLOP/s před výsledky s vektorizací s `-O2`. To je pouhých 3,5% z teoretického maximálního výkonu, což je opravdu málo, ale jak bylo uvedeno výše, Linpack vysokých hodnot ani dosahovat nemůže.



Obrázek 5.2: Výsledky benchmarku Linpack ve dvojité přesnosti pro různé parametry překladače na Odroid XU4. Přesné popisy variant jsou v Příloze A.

Dvojitá přesnost Výsledky ve dvojité přesnosti jsou na Obrázku 5.2. Velikost problému byla zvolena na základě výsledků následujícího testu 600. Při použití `-O1, -O2 a -O3` je vidět výkonový pokles cca o 0,2 GFLOP/s oproti výkonu při použití jednoduché přesnosti. Další parametry upřesňující architekturu poté nehrají téměř žádnou roli a nemá cenu se jimi zabývat. Stejně tak parametry využívající SIMD vektorizační jednotku NEON. Je to proto, že na architektuře ARMv7-a NEON vůbec nepodporuje operace ve dvojité přesnosti[2].

Srovnání s jinými procesory

Hodnota 2,28 GFLOP/s je až nečekaně vysoká, jak je vidět při srovnání s procesory na superpočítačích Anselm a Salomon. Jejich celkový výkon ale spočívá ve vysokém počtu jejich jader. Přidáno je také srovnání s jednodeskovým počítačem RaspberyPi 2, oproti kterému je ARM Cortex A15 asi 10× rychlejší.

Tabulka 5.1: Výkon různých procesorů v benchmarku Linpack [8].

	[GFLOP/s]
Cortex A15@2,0GHz, `gcc5.3 -O3, -mfpu=neon-vfpv4, -ffast-math`	2,28
Salomon - Intel Xeon E5-2680v3 @ 2.5GHz `gcc5.2 -mavx -O3`	6,86
Intel Xeon E5-2665 @ 2.4GHz `gcc4.9 -mavx -march=native -ffast-math`	5,7
Raspberry Pi 2 - ARMv7a @ 1GHz `gcc4.8 + NEON + fastmath`	0,18

Test jader A15 vs A7

Dále byl srovnáván výkon obou typů obsažených jader - úsporného LITTLE Cortexu A7 a výkonnějšího big Cortexu A15. K překladu testů byly již použity parametry překladače získané minulým testem. Testováno bylo několik velikostí problému, které měly ukázat, zda jsou jednotlivé výsledky stálé, nebo se často mění. Oba typy jader běžely na své maximální frekvenci. Z Obrázku 5.3 lze vidět, že slabší Cortex A7 jádro má asi čtvrtinový výkon silnějšího Cortexu A15 a to jak v single tak i v double-precision. Je také dobře vidět bod, ve kterém začnou nedostačovat cache paměti, a výkon začne padat dolů. U single-precision nastává tento bod podle předpokladu až přibližně u 1,5-2násobně větší velikosti problému, než u double-precision.

5.3 HPL

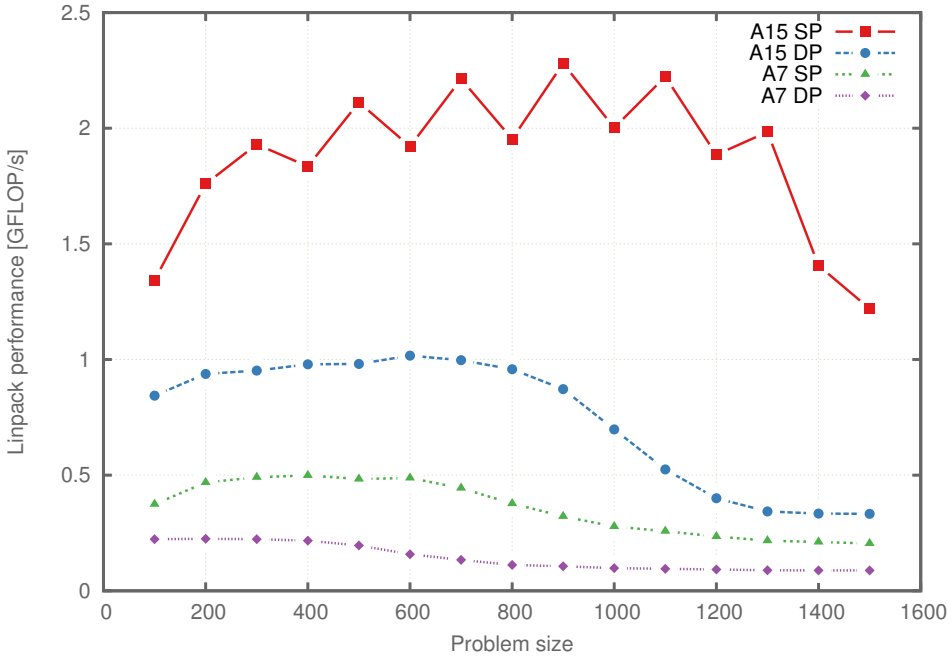
Popis

Původní benchmark Linpack je vhodný pouze pro měření výkonu na jednom jádru jednoho stroje. Dnes se proto využívá hlavně jeho modifikovaná verze HPL – High-Performance Linpack[14]. Ten pomocí technologie MPI propojí dohromady všechny výpočetní uzly a změří jejich výkon dohromady.

Jeho výsledkem je výkon algoritmu v GFLOP/s, který bere do úvahy jak výpočetní výkon jednotlivých uzlů, ale i kvalitu propojení mezi uzly. Je to právě tento benchmark, který je použit pro porovnávání v žebříčku nejvýkonnějších superpočítačů světa Top500. HPL byl použit ve verzi 2.2.

U tohoto benchmarku byla měřena pouze výkonná jádra složená z Cortexů A15, z každého uzlu byla použita všechna 4 jádra. Původně bylo měřeno na frekvenci jader 2000 MHz, ovšem tato frekvence se při testech ukázala jako nestabilní. Stávalo se, že se při delším testu některé kity samovolně restartovaly či se dostávaly do nedefinovaných stavů s chybovými hláškami ohledně integrity datových L1 cache pamětí. Stejné problémy se vyskytovaly u několika kitů.

Tyto problémy byly pravděpodobně způsobeny nedostatečným napájením kitů při vysoké zátěži. Projevovalo se to např. tím, že ač právě při frekvenci 2000 MHz by kit potřeboval



Obrázek 5.3: Srovnání výkonu jader Cortex A15 a Cortex A7 v benchmarku Linpack.

chladit, na větráky se již nedostávalo energie, a byly nuceny běžet na menší otáčky. Výjimkou nebylo napětí snížené na hodnoty okolo 4,2 V. Při testech spotřeby s připojeným měřičem se na frekvenci 2000 MHz benchmark nepodařilo vůbec spustit a systém vždy havaroval. Maximální frekvence proto byla u tohoto testu často snížena na 1800 nebo 1900 MHz.

Nastavení

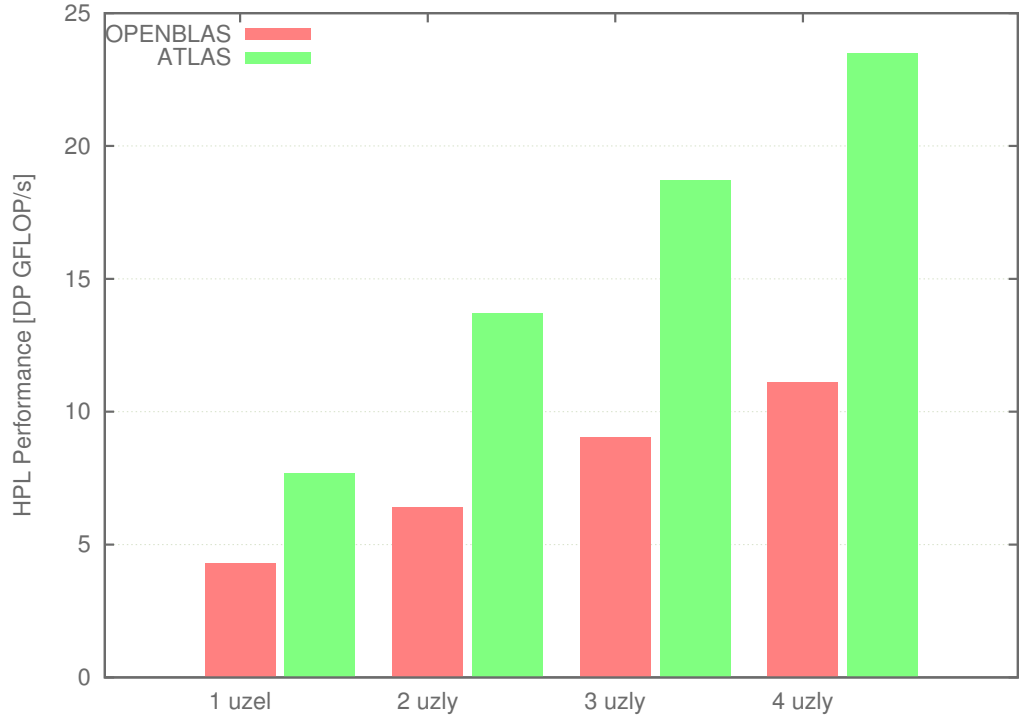
Benchmark HPL má velké množství nastavení, ovládaných konfiguračním souborem HPL.dat. K dosažení co nejlepších výsledků by měl test zabrat co největší porci paměti, ale ta musí být správně zarovnaná do bloků, aby přesahující bloky nezkreslovaly výsledek. Je tedy nutno benchmark vyladit pro danou architekturu.

Klíčové jsou zejména parametry `N, NB, P a Q`. Soubor HPL.dat, se kterým bylo dosaženo nejlepších výsledků pro jeden kit, je přiložen v Příloze B.

ATLAS vs. OpenBLAS

Samotný program benchmarku neobsahuje kód pro výpočty, k tomu jsou využity externí knihovny poskytující rutiny pro lineární algebru, neboli BLAS. Ty bývají při instalaci konfigurovány a optimalizovány pro běh na cílové platformě. Na jejich volbě a vyladění závisí, jak dobrý bude výsledek. Při testování byly vyzkoušeny dvě knihovny – OpenBLAS[1], optimalizovaná pomocí kompilačního softwaru EasyBuild a knihovna ATLAS[2] – přeložená samostatně, mimo EasyBuild. Ty byly proti sobě postaveny a se značným náskokem zvítězila knihovna ATLAS, jak je vidět v grafu na Obrázku 5.4. Ta se ukázala mnohem lépe

[1] http://www.openblas.net/
[2] http://math-atlas.sourceforge.net/



Obrázek 5.4: Srovnání výkonu benchmarku HPL s knihovnami ATLAS a OpenBLAS. Použita byla knihovna OpenMPI 1.10.2 a kit běžel na frekvenci 1800 MHz.

optimalizovaná, a proto byla použita ve všech dalších testech s HPL. Překlad a optimalizace knihovny ATLAS trval na jednom kitu Odroid XU4 téměř jeden den čistého času.

Škálování s frekvencí

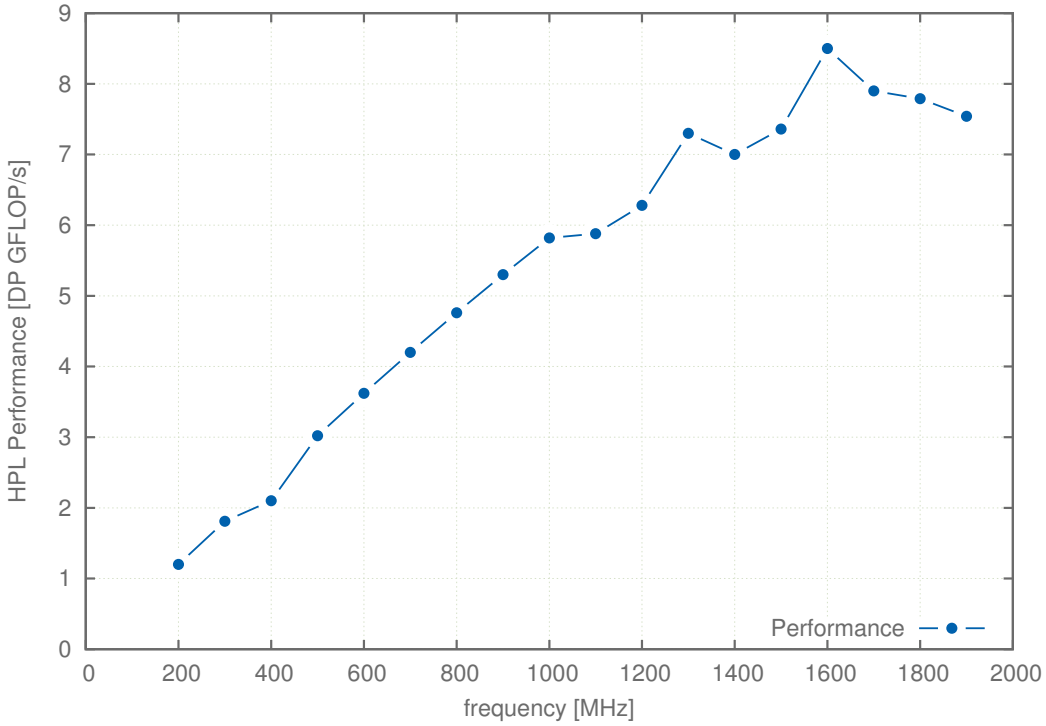
Velmi překvapivým výsledkem dopadlo měření výkonu v GFLOP/s na jednom kitu na jádrech Cortex A15 s rostoucí frekvencí. Intuitivně by se dalo předpokládat, že s rostoucí frekvencí se bude patřičně zvyšovat i získaný výkon, ale není tomu tak, jak je vidět na Obrázku 5.5. Výkonová křivka roste podle očekávání přibližně lineárně až do frekvence 1600 MHz, při které byl naměřen výkon 8,5 GFLOP/s. Od frekvence 1700 MHz a dále se ale začíná výkon snižovat.

Pravděpodobně je to opět způsobeno tím, že při vysokých frekvencích a zátěži má kit příliš velkou spotřebu energie, a na jednotlivé komponenty se nedostává tolik, kolik by potřebovaly. Je otázkou, zda by pro kit nebyl vhodnější silnější zdroj než originální, který má pouze 20W.

8,5 GFLOP/s je velmi dobrým výsledkem, protože při frekvenci 1600 MHz je teoretický výkonový strop v double-precision pouze $16 \cdot 1600/2000 = 12,8$ GFLOP/s, takže v testu bylo dosaženo přibližně 2/3 teoretického výkonu.

Škálování s počtem uzlů

Samozřejmě by šlo dalším laděním parametrů dosáhnout ještě o něco lepších výsledků, co je však důležité, je spíš celkové chování clusteru s přibývajícím počtem uzlů. Z grafu na



Obrázek 5.5: Porovnání výkonu benchmarku HPL běžícího na všech čtyřech jádrech Cortex A15 na jednom kitu Odroid XU4 pro různé frekvence.

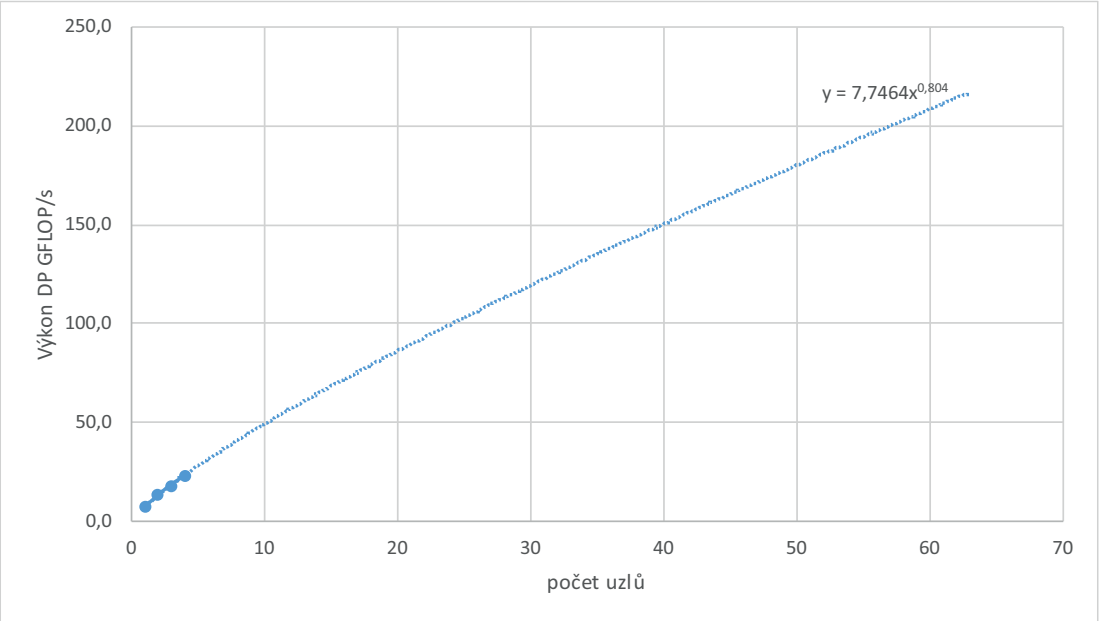
Obrázku 5.4 by se mohlo zdát, že výkon roste lineárně s přibývajícím počtem uzlů, ale není tomu tak. Díky potřebě komunikace a výměně dat s ostatními uzly další uzel nikdy nepřidá 100% svého výkonu a přidaný výkon je s každým dalším uzlem menší.

Na Obrázku 5.6 je zobrazena predikce růstu výkonu s přibývajícím počtem uzlů, vytvořená na základě hodnot známých pro 1–4 uzly. Jako nejlepší varianta se jeví varianta proložení známých hodnot mocninnou řadou $y = 7.7476x^{0.804}$. Pro přesnější predikci by samozřejmě bylo zapotřebí měření s větším počtem uzlů.

Díky tomu je možné odhadnout, kolika uzlů je zapotřebí, aby bylo možné nahradit jeden uzel superpočítače Anselm. Jestliže jeden uzel superpočítače Anselm dosáhl výkonu 215 GFLOP/s, tak by bylo zapotřebí přibližně 63 kitů Odroid XU4. Do úvahy však není bráno, jak by se chovalo takové množství kitů na síti, ani jak by byly fyzicky propojeny.

Poznámka

Co je oproti klasickému Linpacku důležité je, že HPL měří výsledky pouze v double precision, tedy ve dvojité přesnosti s typem `double`. Pro architekturu ARMv7-a použitou v Odroidu to také znamená, že nelze benefitovat ze SIMD vektorových rozšíření ARM NEON, protože ta podporují pouze počítání s typem `float`[2]. Toto můžeme vidět již ve výsledcích z klasického Linpacku, kdy zapnutí vektorizace nevedlo k lepším výsledkům. V tomto mají x86-64 clustery výhodu. Jako doporučení při stavbě budoucího clusteru je tedy nepoužívat 32bitovou architekturu ARMv7-a, ale 64bitovou AArch64, která již plně podporuje jak `float`, tak `double` pro NEON a navíc je plně kompatibilní s IEEE 754[3]. Tato volba by mohla přinést až dvojnásobné navýšení výkonu!



Obrázek 5.6: Predikce růstu výkonu Odroid XU4 clusteru s přibývajícími uzly v benchmarku HPL.

5.4 STREAM

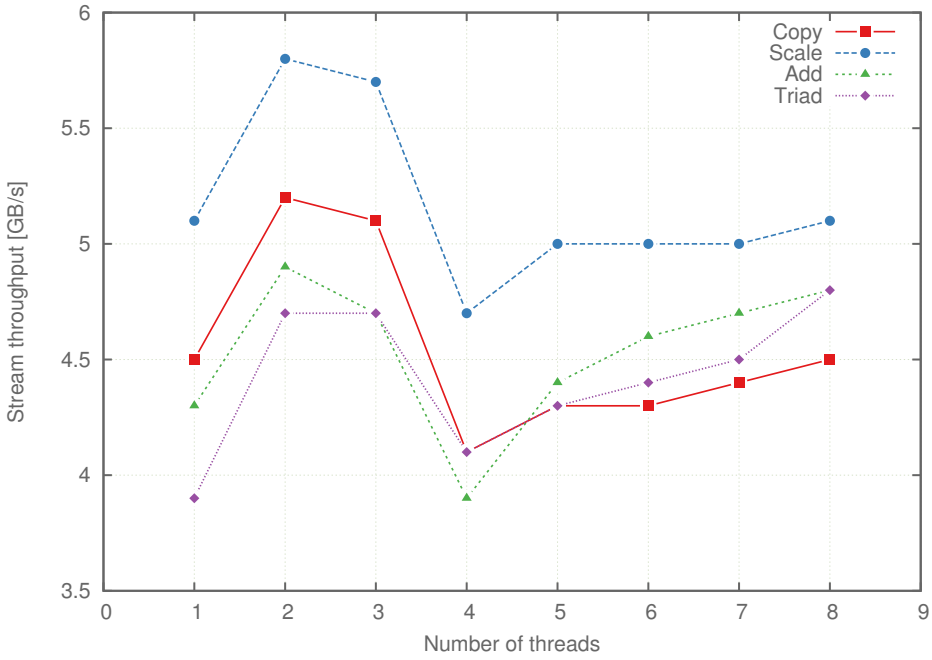
Popis

Benchmark STREAM se zaměřuje na zjištění paměťové propustnosti systému, tedy jakou rychlostí mohu číst a zapisovat data z hlavní paměti. Jeho výsledkem je datová propustnost v GB/s. Načítaná data musejí mít takovou velikost, že se nesmějí vejít do pamětí cache, abychom mohli zjistit propustnost hlavní paměti. Nad získanými daty benchmark provádí základní vektorové operace, kde a, b a c jsou data načítaná z paměti a x je konstanta, kterou není nutné načítat z hlavní paměti. Všechny operandy jsou typu `double` – tedy mají velikost `8B`. Operace `Copy` pouze načte operand z paměti a zapíše jej zpět, tedy celkem přispěje `16B` do celkové paměťové propustnosti.

- operace `Copy` $a = b$
- operace `Scale` $a = x * b$
- operace `Add` $a = b + c$
- operace `Triad` $a = b + x * c$

Benchmark pomocí knihovny OpenMP umožňuje spustit několik vláken se svými instancemi. Více vláken je potřeba k tomu, aby se dalo určit, jaký nejmenší počet vláken je potřeba k tomu, aby byl naplno využit potenciál paměti. Této znalosti lze poté využít při psaní aplikací na míru danému hardwaru. Pro přiřazení OpenMP vláken k jádrům byly užity příkazy `export OMP_NUM_THREADS=4` a `export GOMP_CPU_AFFINITY=4,5,6,7`.

Výsledky se vždy zakládají na několika opakovaných měřeních, ze kterých byl spočítán aritmetický průměr.



Obrázek 5.7: Výsledky benchmarku STREAM na Odroidu XU4. Při 1–4 vláknech byla použita jádra Cortex A15 a při 5–8 vláknech byla přidávána jádra Cortex A7.

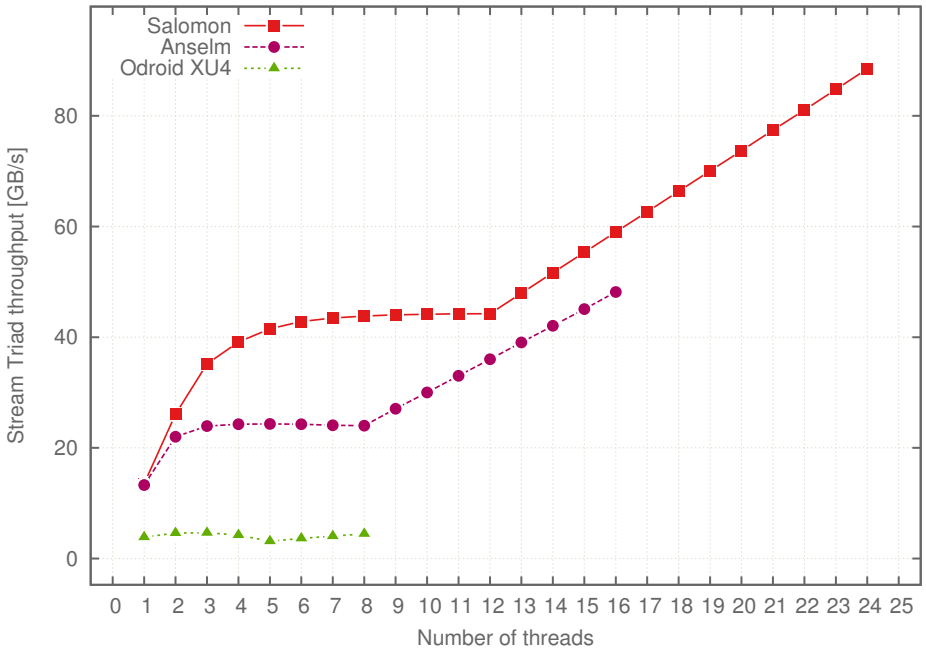
Výsledky pro Odroid XU4

Výsledky změřené na jednom kitu Odroid XU4 je možné vidět na Obrázku 5.7, který ukazuje závislost počtu vláken na výsledné propustnosti. Proběhlo 10 běhů a výsledky jednotlivých běhů si byly velmi podobné. Je vidět, že nejlepší výsledná propustnost je u všech testů při použití dvou vláken. Použití více než dvou vláken již nevede k lepším výsledkům. Při použití 3 a 4 vláken propustnost klesá. Je to zřejmě tím, že si vlákna konkurují při přístupu ke společné L2 cache. Při přidání vláken z úsporného clusteru je vidět nárůst výkonu způsobený pravděpodobně tím, že úsporná jádra mají také svoji L2 cache. Také si tolik nekonkurují, protože mají přibližně 4× menší výkon. Jako závěr je tedy možné shrnout, že se nevyplatí používat více vláken pro paralelní načítání z paměti.

$$825\ \texttt{MHz} \cdot \frac{2\ \texttt{přístupy}}{\texttt{takt}} \cdot \frac{2\ \texttt{kanály} \cdot 32\texttt{b šířka rozhraní}}{8\ \texttt{bitů na Byte}} = 13,2\ \texttt{GB/s} \tag{5.3}$$

Výsledky se bohužel neblíží k teoretické možné výkonnosti vypočtené v rovnici 5.3, ale dosahují pouze na 5,5 GB/s. Tyto hodnoty bohužel potvrzuje i benchmark STREAM ve variantě pro OpenCL na grafickém akcelerátoru – GPU-STREAM[17], který podával nejlepší propustnost 7,5 GB/s při variantě `Triad`, což je pouze o přibližně 2 GB/s lepší výsledek.

Test pro grafický akcelerátor je možné využít proto, že jak obě CPU, tak i GPU využívají jednu a tu samou operační paměť. Tato vlastnost by mohla přinést zajímavou výhodu oproti velkým grafickým kartám, které musejí kopírovat data z hlavní paměti a zpět.



Obrázek 5.8: Výsledky benchmarku STREAM na kitu Odroid XU4 a jednom uzlu superpočítačů Salomon a Anselm

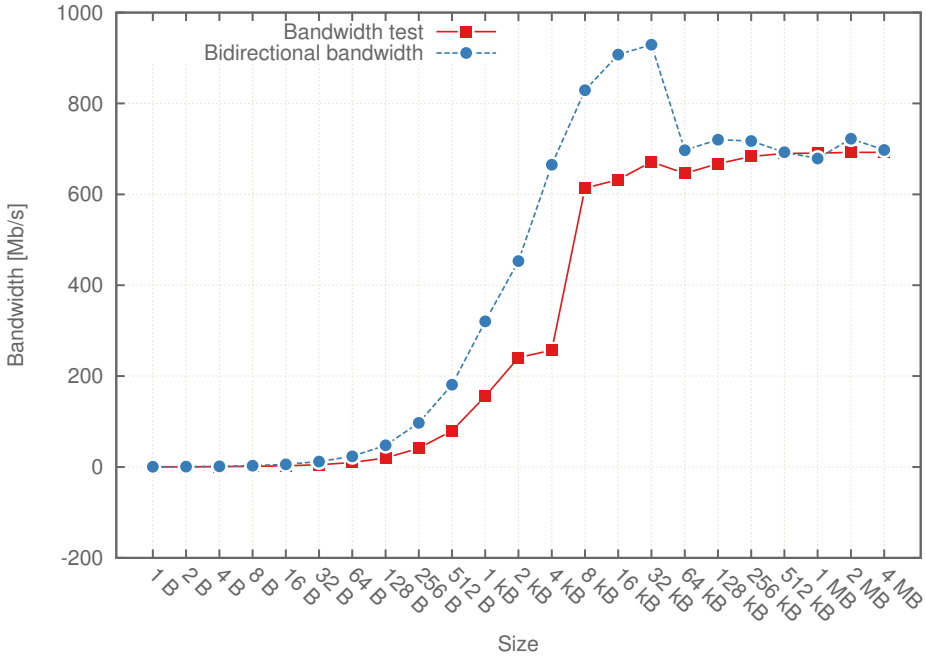
Srovnání se superpočítači

Pokus na jednom uzlu superpočítače Salomon z Obrázku 5.8 byl spuštěn celkem 10× a výsledky jednotlivých běhů byly téměř identické. Každý uzel na Salomonu má dva sockety a s každým socketem i NUMA uzel, na kterých je nainstalováno 64GB paměti. V grafu jde na první pohled vidět, které testy běžely pouze na jednom (1–12 vláken) a které již na více socketech (13–24 vláken). Od jednoho do dvanácti vláken výkon roste, ovšem od šestého vlákna dále je přínos každého dalšího vlákna téměř nulový, protože propustnost je již na dosažitelném maximu – 44,2 GB/s na procesor. To jsou ale pouze 2/3 z teoretického maxima 68 GB/s. Při 13 a více vláknech hodnoty opět začínají růst a to téměř přesně lineárně, protože tyto vlákna již běží na druhém socketu a přistupují do jeho paměti. Druhá část není tak skoková jako první, protože výpočet je již rovnoměrně rozložen mezi více jader. Druhý socket má téměř identickou propustnost jako první – 44,2 GB/s. Při porovnání s kitem Odroid XU4 je vidět, že ve špičce má Odroid přibližně 15× menší paměťovou propustnost, než jeden uzel na superpočítači Salomon.

Totéž ale v menším měřítku jde pozorovat i u superpočítače Anselm, také v grafu na Obrázku 5.8. Anselm má také dva sockety s NUMA uzly, ale na každém pouze 8 jader a 32 GB paměti. Výsledkem je, že jeden uzel superpočítače Anselm má asi 8× větší propustnost, než Odroid XU4.

5.5 OSU Micro-Benchmarks

Často používaným prostředkem pro paralelizaci výpočtů na superpočítačových clusterech je rozhraní MPI. Na kitech je nainstalována varianta OpenMPI ve verzi 1.10.2. Ke změření vlastností instalace byly vybrány testy ze sady OSU Micro-Benchmarks, které zahrnují testy



Obrázek 5.9: Výsledky testů `osu_bw` a `osu_bibw` ze sady OSU Micro-Benchmarks. Ukazují propustnost sítě mezi dvěma uzly pro různě velké objemy posílaných dat.

point-to-point a kolektivních komunikací[3].

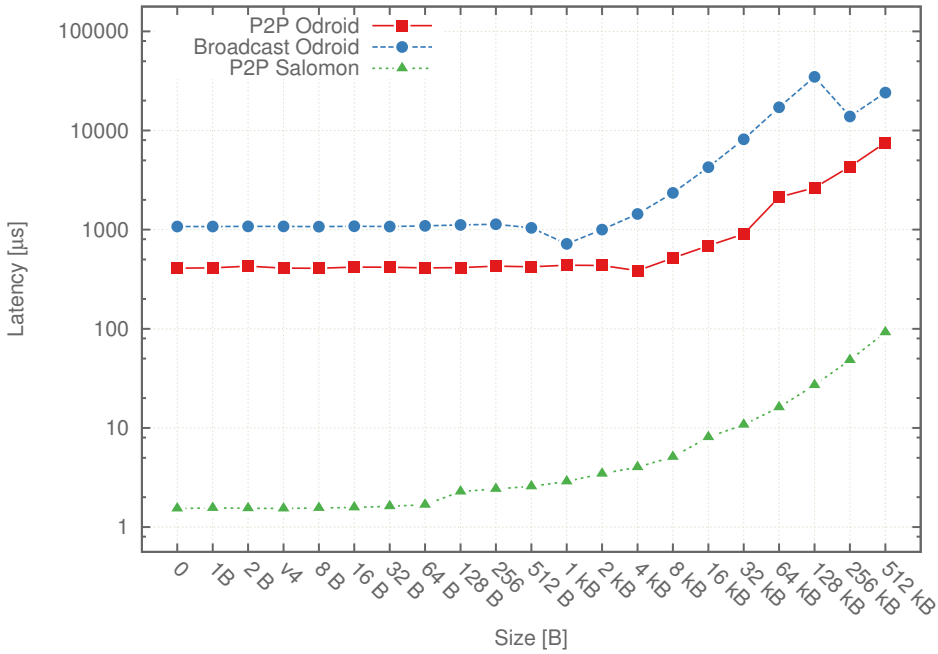
Propustnost sítě Pro zjištění propustnosti byly vybrány dva point-to-point testy, měřící maximální propustnost komunikace mezi dvěma uzly. První byl `osu_bw` (Bandwidth test), ve kterém si dva uzly navzájem posílají zprávy o různé velikosti, a zároveň je měřen čas, za který se zpráva vrátí zpět k odesílateli. Druhý byl test `osu_bibw` (Bidirectional bandwidth test), který se liší od prvního v tom, že oba uzly posílají zprávy zároveň a dokáží tak lépe využít kapacitu linky.

Výsledky obou testů lze nalézt v grafu na Obrázku 5.9. Je vidět, že USB síťová karta na kitech má velké latence, které se plně překryjí až s větším objemem dat, přibližně od 8 kB nahoru. Po překrytí latencí se propustnost drží na stabilní hodnotě okolo 700 Mb/s, což je 70% z teoretické propustnosti gigabitového Ethernetu.

Latence V dalším testu byly srovnány latence point-to-point komunikace a hromadného rozhlášení neboli broadcastu. Pro měření point-to-point byl použit test `osu_latency`, ve kterém se měří délka trvání cesty zprávy z uzlu A do uzlu B a zpět. Broadcast byl měřen testem `osu_bcast`, do kterého bylo zapojeno všech 16 jader ze čtyř uzlů clusteru. U každého testu byly provedeny stovky až tisíce iterací, z nichž byl vytvořen průměr.

Výsledky jsou vidět v grafu na Obrázku 5.10 a potvrzují vysokou latenci síťové karty. Měření na clusteru Salomon se sítí InfiniBand ukázalo přibližně 100× nižší latence, než na Odroid clusteru.

[3] http://mvapich.cse.ohio-state.edu/benchmarks/



Obrázek 5.10: Výsledky testů `osu_latency` a `osu_bcast` ze sady OSU Micro-Benchmarks, ukazující latenci síťových spojení. Osa y je provedena v logaritmickém měřítku.

5.6 Samostatně zvolený problém

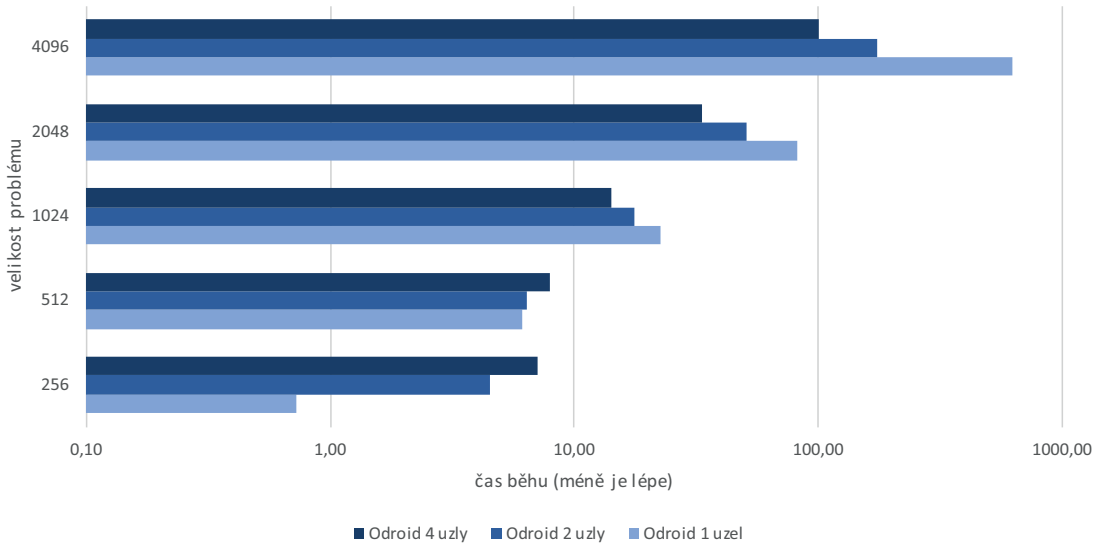
Jedním z bodů práce bylo vybrat vhodný problém a na něm demonstrovat efektivní využití navrženého clusteru. Jako tento problém byl zvolen 3. projekt do předmětu ARC (Architektura a programování paralelních systémů) na VUT FIT. V projektu byl řešen problém nalezení distribuce tepla v ustáleném stavu ve 2D na tenké destičce. Čtvercová destička byla rozdělena na rovnoměrné části mezi jádra, která spolupracují na výpočtu. Ta si pak po každé iteraci vyměňují hodnoty na okrajích svých částí.

Tento projekt je zajímavý, protože má dobrý poměr mezi časem stráveným výpočty a komunikací. Navíc jsou zde zastoupeny jak komunikace point-to-point, tak i kolektivní komunikace jako `MPI_Gatherv()`, `MPI_Scatterv()` a redukce `MPI_Allreduce()`.

Svoje původní řešení tohoto projektu jsem upravil tak, aby bylo vhodné k měření výkonnosti jak na kitech, tak i na superpočítači a to tak, že jsem nastavil počet provedených iterací algoritmu na fixní hodnotu 5000. Velikost problému zde zastupuje velikost hrany hlavní (čtvercové) destičky. Velikosti problému byly zvoleny 256, 512, 1024 a 2048.

Využívání více uzlů s sebou pochopitelně nese i nutnou režii k jejich synchronizaci. Účelem následujícího testu bylo zjistit, od jakého poměru komunikace k výpočtům v aplikaci se vyplatí pro výpočty použít více uzlů.

Výsledek Výsledky jsou v grafu na Obrázku 5.11. V něm jsou zaneseny celkové časy běhů pro 5000 iterací algoritmu na Odroidech, kratší čas samozřejmě znamená lepší výsledek. V grafu lze pozorovat, že pro velmi malé velikosti problému (256) vítězí suverénně konfigurace s jedním uzlem, protože komunikace srazí výkon u více-uzlových řešení. Hraničním bodem je destička o délce hrany 512, kdy jsou časy všech řešení vyrovnané.



Obrázek 5.11: Srovnání času běhů ARC projektu na různém počtu uzlů Odroidu XU4. Slouží k určení, od jaké velikosti problému se vyplatí využít více uzlů Odroidu XU4. Osa x je provedena v logaritmickém měřítku.

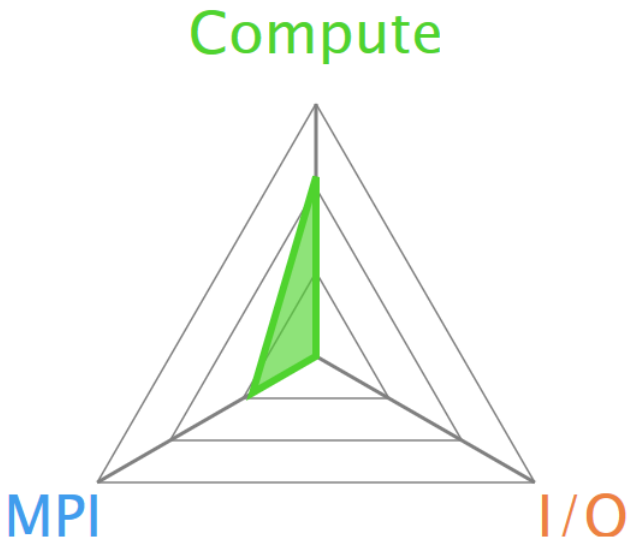
Právě tato hraniční varianta byla analyzována v softwaru Allinea tools na clusteru Salomon a bylo zjištěno, že aplikace tráví 70% běhu výpočty a 30% času v MPI rutinách, jak je možné vidět v grafu na Obrázku 5.12. Z tohoto můžeme tedy usuzovat, že aby se více-uzlové aplikace vyplatily, je potřeba, aby výpočet zabral alespoň 70% času běhu aplikace.

Porovnání se superpočítačem

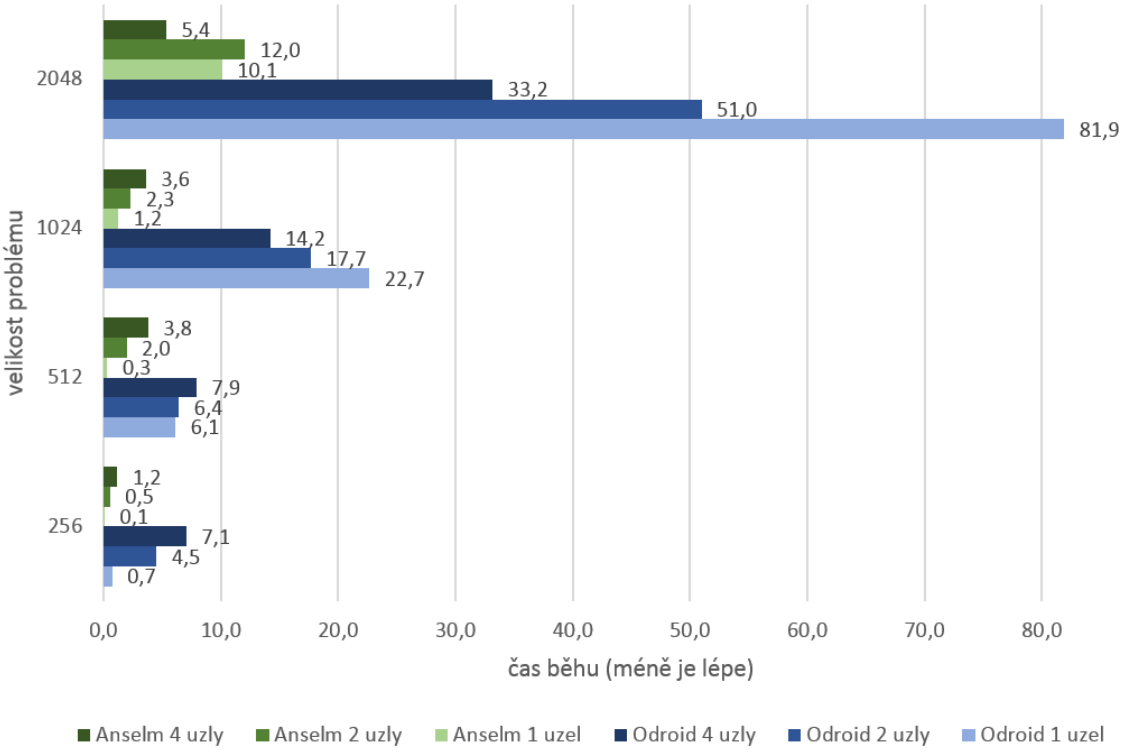
Ten samý test byl prováděn na superpočítači Anselm, kde byl opět zvolen fixní počet 5000 iterací a měřen čas, za který se těchto 5000 iterací vykonalo. Cílem bylo zjistit, kolikrát je rychlejší superpočítač, než postavený cluster.

Na Anselmu byl projekt přeložen pomocí toolchainu `gompi/2015g`, který obsahuje GNU-/GCC ve verzi 4.9.3 a OpenMPI 1.8.8. Vše bylo překládáno pouze s přepínači `-std=c++11 -W -Wall -Wextra -g -pedantic -O3`.

Ve srovnání celkem bez překvapení zvítězil superpočítač, ale jeho vítězství nebylo zdaleka tak drtivé jako při benchmarku HPL. Jak je vidět v grafu na Obrázku 5.13 jeden uzel na Anselmu je přibližně 8× rychlejší, než jeden uzel Odroid clusteru. Na to, že jeden uzel na Odroidu má pouze 4 jádra je to velmi dobrý výsledek, protože to znamená, že jedno jádro Odroidu bylo pouze 2× pomalejší, než jedno jádro na Anselmu.



Obrázek 5.12: Srovnání času stráveného v ARC projektu 3: výpočty, MPI komunikacemi a vstupně-výstupními operacemi při hraniční velikosti problému 512.



Obrázek 5.13: Srovnání časů běhů ARC projektu na uzlech Odroid clusteru a superpočítače Salomon.

Kapitola 6

Energetická spotřeba

Tato část byla zaměřena na spotřebu kitu a na jeho odběr proudu. Naměřené charakteristiky kitů byly po naměření vyhodnoceny a porovnávány s clusterem Anselm.

6.1 Metodika měření

Jednotka použitá při porovnávání v testech je efektivita, měřená v GFLOP/s/W. Ta říká, jakého výkonu operací v plovoucí řádové čárce za jednu sekundu je schopno dané zařízení dosáhnout na jeden Watt příkonu. Jinými slovy porovnáváme poměr vykonané práce v GFLOP a práce v Joulech, která k tomu byla zapotřebí, jak je vidět v Rovnicích 6.1 a 6.2 (protože Watt je definován jako je práce za čas).

Odroid XU4

Na rozdíl od svého předchůdce, Odroidu XU3, XU4 již nemá zabudované obvody pro softwarové měření spotřeby přímo na kitu. Měření tedy bylo nutné provádět fyzicky za pomoci měřících přístrojů.

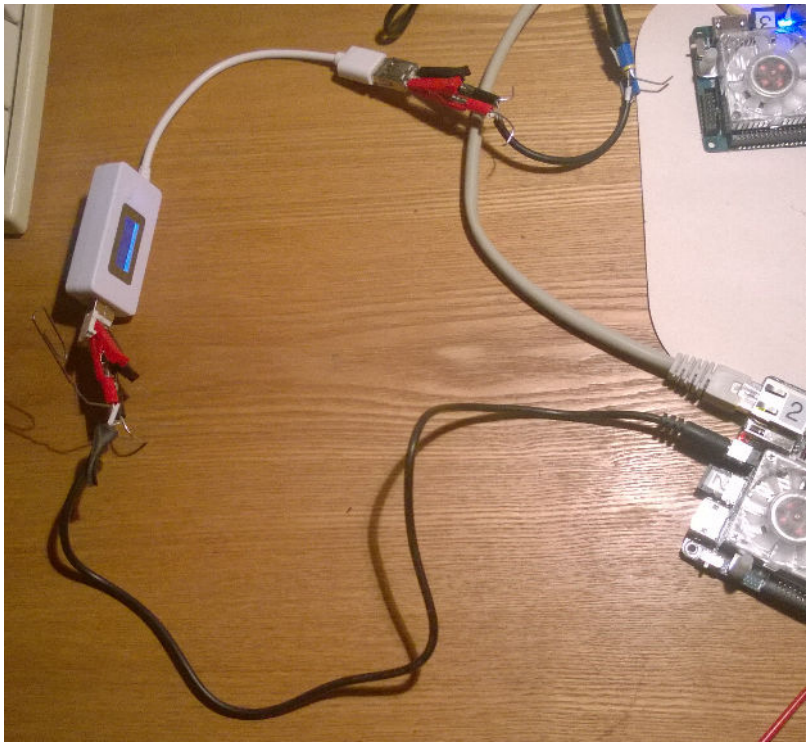
To by šlo samozřejmě provést nejjednodušeji pomocí měřiče do zásuvky, ale nakonec byla zvolena metoda měření až za zdrojem. Je to proto, že zdroj není dokonalý a projevují se na něm určité ztráty, což by mohlo zanášet chyby, a nešly by rozumně porovnávat výsledky na kitu s výsledky naměřenými na superpočítači.

Měření bylo provedeno pomocí měřidla KCX-017 podle zapojení viditelného na fotografii na Obrázku 6.1. Toto měřidlo poskytuje jednak informace o aktuálním proudu protékajícím obvodem a napětí, ale má i paměť, ve které uchovává informace o spotřebované proudu kapacitě v mAh. K výsledné efektivitě se lze dostat pomocí Rovnice 6.3.

$$[\mathrm{W}] = \left[\frac{\mathrm{J}}{\mathrm{s}}\right] \tag{6.1}$$

$$\left[\mathrm{GFLOP} \cdot \mathrm{J}^{-1}\right] = \left[\frac{\mathrm{GFLOP}}{\mathrm{J}}\right] = \left[\frac{\mathrm{GFLOP}}{\mathrm{mAh} \cdot 10^{-3} \cdot 3600 \cdot \mathrm{V}}\right] \tag{6.2}$$

$$\left[\mathrm{GFLOP} \cdot \mathrm{s}^{-1} \cdot \mathrm{W}^{-1}\right] = \left[\frac{\mathrm{GFLOP/s}}{\mathrm{W}}\right] = \left[\frac{\mathrm{GFLOP/s}}{\mathrm{mAh} \cdot 10^{-3} \cdot \mathrm{V} \cdot \mathrm{h}^{-1}}\right] \tag{6.3}$$



Obrázek 6.1: Fotodokumentace postupu měření spotřeby u Odroidu XU4.

superpočítač Anselm

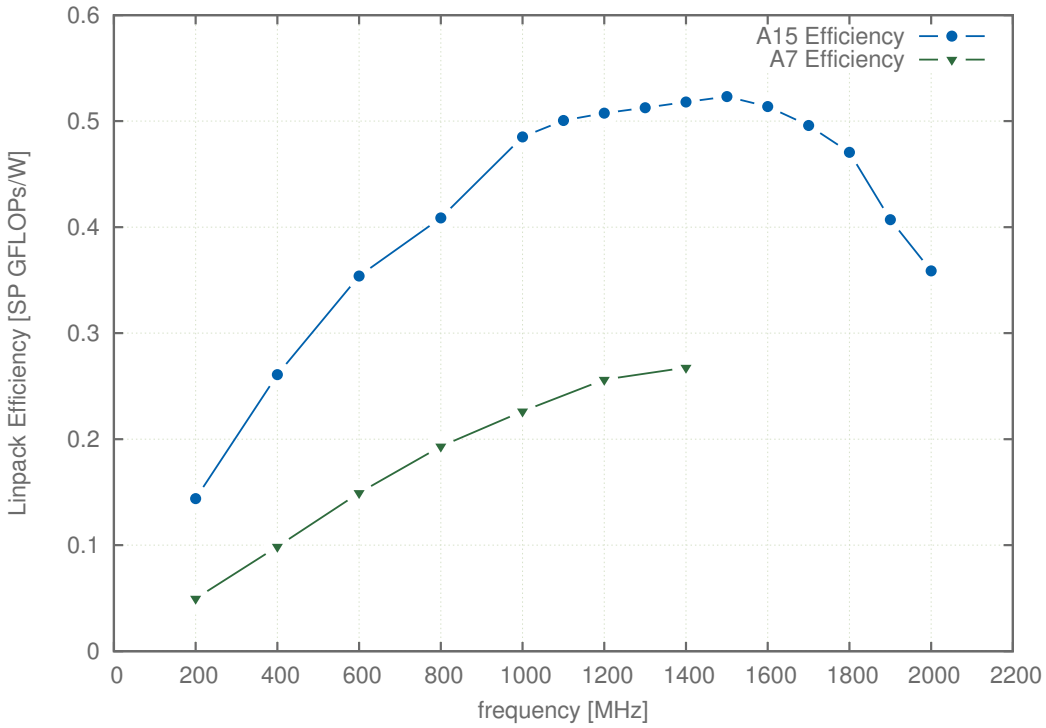
Na superpočítači Anselm bylo měření prováděno pomocí utility `pcm-power` z modulu `intelpcm` (Intel performance counter monitor), která využívá vnitřní čítače procesoru bez nutnosti jakkoliv fyzicky zasahovat do hardwaru[13].

Výstupem `pcm-power` je jednak celková spotřebovaná energie v Joulech, ale i průměrný příkon ve Wattech. Právě ten je využit v porovnávacích testech. Dokonce od sebe umí oddělit příkon čipsetu samotného a příkon pamětí, ale tyto dva údaje jsou v testu sečteny dohromady.

6.2 Klidové hodnoty

V klidových podmínkách, kdy není zatížen žádnou prací, má kit spotřebu 0,25 A což dělá příkon okolo 1,125 W. To je při situaci, kdy je kit připojen do sítě, ale nemá žádné aktivní připojení přes SSH, ani nemá připojen síťový disk. Také mají všech jádra nastaven governor *interactive*, kde se frekvence mění se zátěží.

Po připojení síťového disku přes NFS a jednom SSH připojení stoupne odběr na hodnoty okolo 0.5 A, což dává při napětí 5 V příkon asi 2,5 W.



Obrázek 6.2: Efektivita kitu v GFLOP/s/W při benchmarku Linpack v jednoduché přesnosti. Zatíženo bylo pouze jedno jádro.

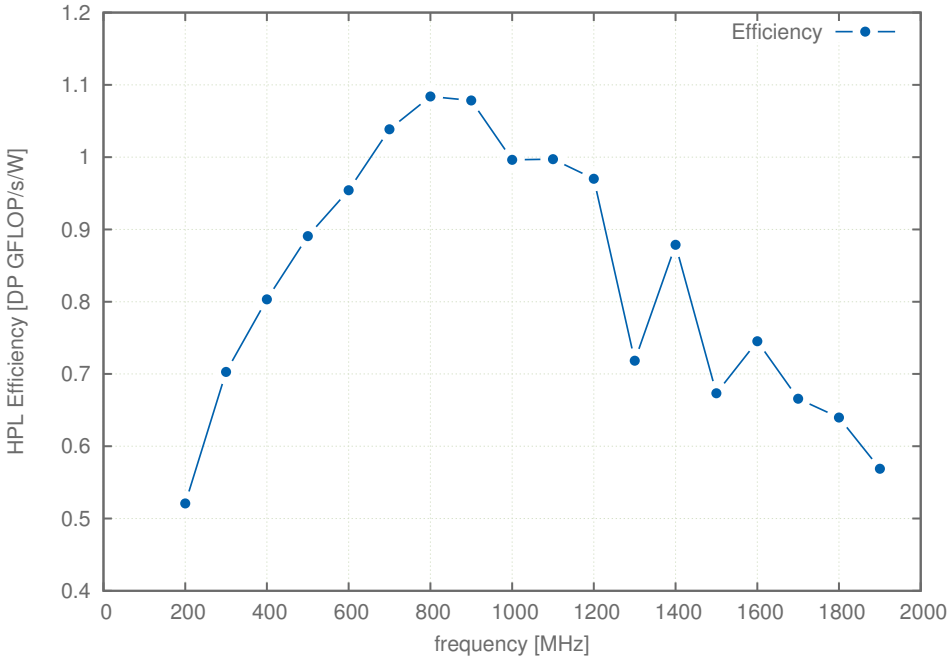
6.3 Efektivita GFLOP/s/W

6.3.1 Škálování efektivity s frekvencí

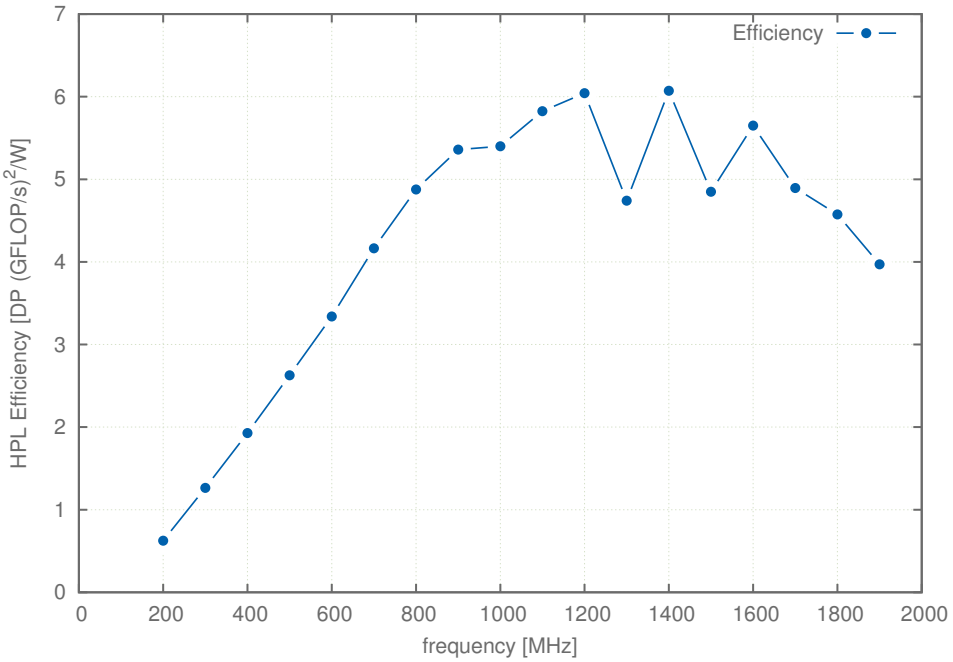
Cílem tohoto testu bylo zjistit, jak roste příkon kitu se změnami frekvence, a také nalézt ideální operační frekvenci kitu, při které dává nejlepší výsledky při zachování co nejmenšího odběru. V tomto testu byla vždy napevno nastavována frekvence kitu a s touto byla měřena výkonnost pomocí benchmarků Linpack při použití jednoho jádra a benchmarku HPL při použití Cortex A15 jader z jednoho uzlu. Ostatním jádrům byla nastavena co nejmenší frekvence, aby nezkreslovala výsledky. Aby bylo možné nastavovat frekvenci, je potřeba daná jádra přepnout na governor `performance`. Frekvence na Odroidu jdou nastavovat vždy po kroku o velikosti 100 MHz. U úsporného Cortexu A7 je to od 200 MHz do 1400 MHz a u výkonnějšího Cortexu A15 od 200 MHz do 2000 MHz.

Linpack Nejprve byla testována efektivita při benchmarku Linpack na jednom jádru kitu. Zkoušena byla obě jádra – úsporné i výkonné. Výsledky tohoto testu jsou k nalezení v grafu na Obrázku 6.2 a ukazují, že nejlepší operační frekvence z hlediska efektivity jsou mezi 1–1,6 GHz. Dále také výsledky ukazují, že slabší jádra Cortex A7 nejsou vůbec vhodná pro výpočty, ale pouze pro úsporný běh, kdy se nic nepočítá.

Hodnoty čistých výsledků okolo 0,5 GFLOP/s/W nejsou příliš dobré, ale je třeba mít na paměti, že u nich se celá spotřeba kitu, včetně režie, rozpočítá pouze jednomu jádru.



Obrázek 6.3: Efektivita kitu Odroid XU4 v GFLOP/s/W při benchmarku HPL. Pracovala všechna 4 jádra Cortex A15.



Obrázek 6.4: Efektivita kitu Odroid XU4 v $(\mathrm{GFLOP/s})^2/\mathrm{W}$ při benchmarku HPL. Pracovala všechna 4 jádra Cortex A15.

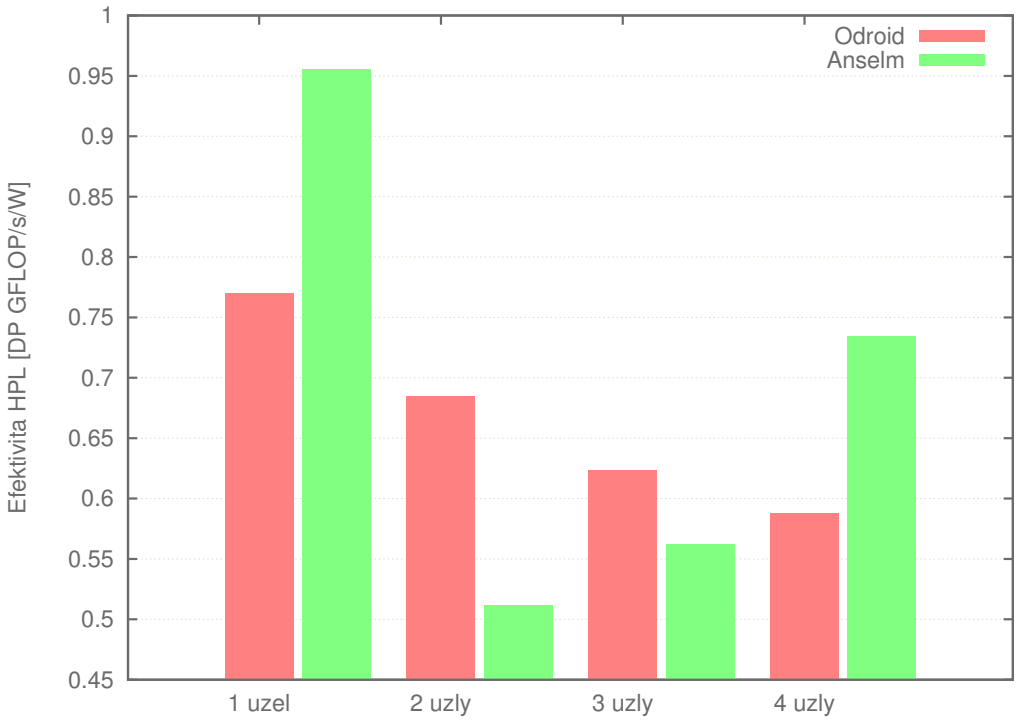
HPL V dalším testu efektivity byl změřen benchmark HPL. Výsledky testu jsou k dispozici v grafu na Obrázku 6.3. Díky použití optimalizovaných rutin pro BLAS jsou jeho

výsledky výrazně výše než u Linpacku. Nejlepší výsledek okolo 1,1 GFLOP/s/W ukazují výsledky při frekvenci 800MHz, což ovšem není frekvence, na které by měly běžet kity v reálném clusteru.

Proto byl sestaven graf, ve kterém hrají větší roli výkon v GFLOP/s, který je umocněn a započítán jako $(\text{GFLOP/s})^2/\text{W}$. Takovýto graf dává lepší představu o tom, na které frekvenci by měly kity ideálně běžet a je k vidění na Obrázku 6.4. Z něj vyplývá, že ideální frekvence k běhu jsou 1200, 1400 a 1600 MHz. Není proto divu, že na frekvenci 1600 MHz dosahuje kit nejlepšího výkonu, jak je viděti na Obrázku 5.5 a proto bych ji doporučil jako vhodnou pracovní frekvenci pro reálně použití. Při ní má kit efektivitu přibližně 0,75 GFLOP/s/W.

6.3.2 Porovnání se superpočítačem

Další test byl zaměřen na porovnání se superpočítačem Anselm. Porovnávanou veličinou byla opět efektivita v GFLOP/s/W při testu HPL. Aby byla započtena režie na komunikaci u obou testovaných subjektů, použil jsem vždy 1, 2, 3 a 4 uzly, které využívaly všechna svoje jádra. Výsledky jsou vidět na Obrázku 6.5. Procesory Intel Xeon E5-2665 běžely téměř celou dobu testu velmi blízko svého TDP, většinou mezi 100–105 W. Při využití všech čtyř uzlů má Odroid cluster efektivitu přibližně 0,58 GFLOP/s/W.

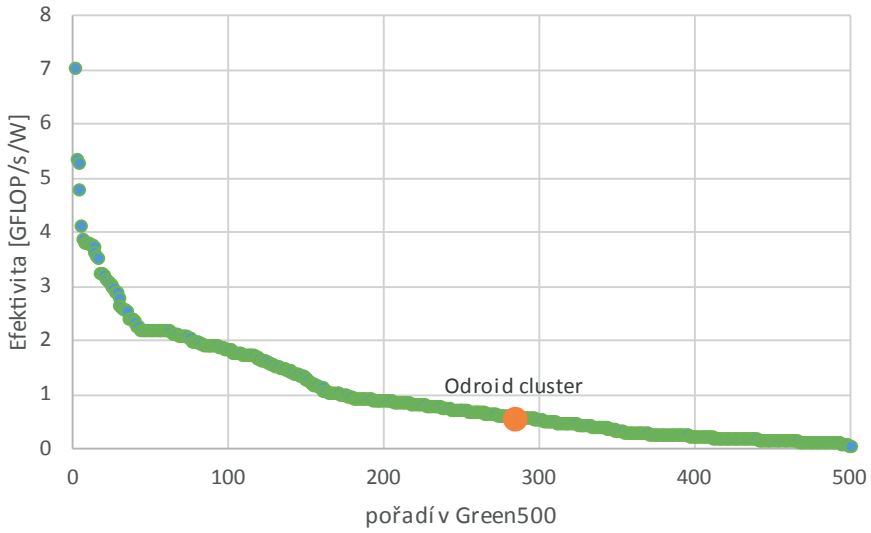


Obrázek 6.5: Škálování efektivity v GFLOP/s/W se zvyšujícím se počtem uzlů na kitech Odroid XU4 a na superpočítači Anselm při benchmarku HPL. U Odroidu XU4 pracovala na jednom uzlu všechna 4 jádra Cortex A15.

6.4 Green500

Green500 je celosvětovým žebříčkem superpočítačů, který je řadí podle celkové efektivity všech uzlů v benchmarku HPL měřené v GFLOP/s/W.

Podle nejnovějšího žebříčku zveřejněného v listopadu 2015 by se postavený cluster umístil se svými 0,58 GFLOP/s/W při čtyřech uzlech na 285. místě, jak je vidět v žebříčku vizualizovaném na Obrázku 6.6[7]. Je však potřeba mít na paměti, že v žebříčku jsou clustery, které mají řádově desítky až stovky tisíc jader a efektivita Odroid clusteru by šla s větším počtem uzlů ještě níže.



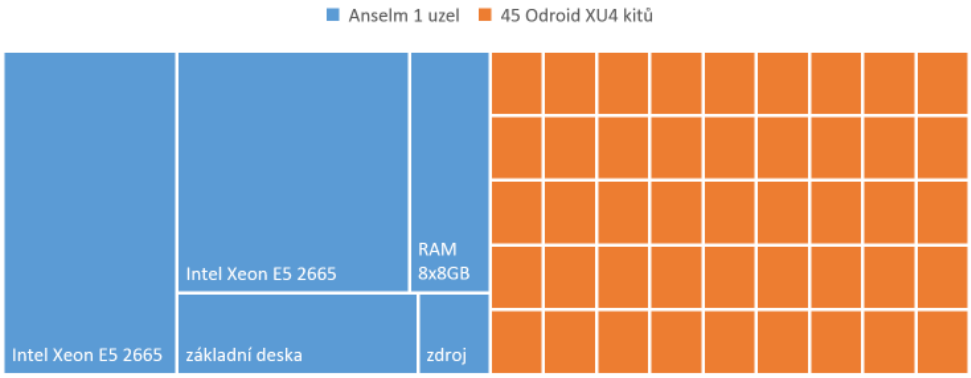
Obrázek 6.6: Žebříček Green500 s vyznačeným bodem, kde se nachází Odroid cluster. Každý bod značí jeden superpočítač z žebříčku.

6.5 Porovnání pořizovací ceny

Nakonec byly srovnány pořizovací náklady jednoho uzlu clusteru Anselm a potenciálního clusteru složeného z Odroidů XU4, jejichž množství muselo být takové, aby si clustery výkonově odpovídaly. Toto srovnání je samozřejmě dosti zjednodušené, protože nezahrnuje náklady na pořízení potřebné infrastruktury a práci, ale poskytuje alespoň rámcový odhad, zda by se stavba tohoto potenciálního clusteru finančně vyplatila. Dále je také zanedbán fakt, že při odběru většího množství jednotek by se ceny za kus pravděpodobně snížily.

Ceny za kity Odroid XU4 byly brány z oficiálních stránek společnosti Hardkernel, podle kterých je možné jeden kit pořídit za $74[10] a 16GB microSD kartu za dalších $14, což dělá dohromady $88 za jeden kit.

Skutečná pořizovací cena za jeden uzel Anselmu, Bullx Blade 510, bohužel není veřejně známá. Cena byla proto spočtena pouze přibližně ze známých komponent. Podle Intel ARK je doporučená cena pro Intel Xeon E5-2665 $1440[11], avšak do uzlu jsou potřeba dva, takže celkem $2880. Dále připočteme $500 za operační paměti v konfiguraci 8x8GB, DDR3 a ECC. Dále také $500 za serverovou základní desku se dvěma sockety LGA 2011 + přibližně $150 za



Obrázek 6.7: Grafické porovnání přibližné ceny komponent jednoho uzlu clusteru Anselm s kity Odroid XU4 mající stejnou celkovou cenu.

adekvátní zdroj. Ceny byly získány jako průměr cen vyhovujících komponent v internetovém obchodě Newegg[1]. Celková suma za jeden uzel je odhadem přibližně $4000.

Za cenu jednoho uzlu superpočítače Anselm je tedy možné pořídit přibližně 4000/88 = 45 kitů Odroid XU4. Tento poměr je graficky znázorněn na Obrázku 6.7. Avšak podle predikce v Kapitole 5.3 je potřeba přibližně 63 kitů k vyrovnání výkonnosti, takže výhodněji vyjde jeden uzel superpočítače Anselm.

[1] http://www.newegg.com/

Kapitola 7

Vhodnost k užití v praxi

V této kapitole bych chtěl zúročit znalosti a zkušenosti nabyté prací s kity Odroid XU4 pro rozhodnutí, zda jsou vhodné pro stavbu potenciálního budoucího clusteru. Zhodnocení bych rád rozdělil do několika části, a to doporučení ohledně samotného kitu Odroid XU4, architektury ARMv7-a a architektury ARM obecně.

Odroid XU4 - doporučení - NE

- – Kit používá ne zcela vyladěný systém - některé funkce se teprve dodělávají. Např. podpora pro čítače procesoru musela být dodána do jádra manuálně.
- – Často se stává, že se kit po restartu nedostane do stavu, kdy je možné s ním navázat spojení přes SSH, kvůli problémům s gigabitovým Ethernetem. Toto možná bude opraveno v jádře.
- – Kit nezvládá provoz na frekvenci 2 GHz a při náročnějších aplikacích na ní není stabilní.
- – Kit nemá zabudované sensory na měření spotřeby.
- – Vývoji se věnuje málo lidí a nestíhají se starat o systémy pro několik vydaných kitů.
- – Samsung neposkytuje technickou dokumentaci k čipům Exynos.
- – Odpovídající výkon v kitech lze pořídit za větší cenu, než jeden uzel superpočítače.
- \+ Kit má dobrou podporu na fórech od komunity.
- \+ Kit má celkem dobrý výkon.
- \+ Deska má malé rozměry a neobsahuje zbytečnosti jako WiFi či Bluetooth.
- \+ Možnost vyměnit microSD karty za přibližně 4× rychlejší MMC moduly.

ARMv7-a - doporučení - NE

- – Vektorová jednotka NEON nepodporuje výpočty ve dvojité přesnosti a u těch v jednoduché přesnosti porušuje normu IEEE 754.
- – V jádře systému není podpora pro technologii NUMA a proto v MPI nefungují některé příkazy ohledně připínání vláken k jádrům.

– Podpora pro 32bitové architektury není u HPC softwaru nejlepší. Hlavně u programu EasyBuild pro kompilaci a správu HPC softwaru.

\+ Procesory Cortex A15 mají oproti svým předchůdcům velký výkonový náskok.

\+ Architektura ARMv7-a má již dobrou podporu v překladači GNU/GCC a systému GNU/Linux obecně.

ARM - doporučení - ANO Většina zařízení postavených na procesorech s architekturou ARM jsou vývojářské desky a jsou na to zaměřeny, tedy jsou vybaveny různými GPIO a podobnými piny. Často jsou to čipy původně určené pro trh s chytrými telefony (jako Exynos 5422 použitý v Odroidu) a jsou heterogenní – mají v sobě několik typů jader a často v sobě nosí podporu pro funkce mobilního telefonu, včetně služeb v systému.

K většímu prosazení v oblasti HPC by se musel najít výrobce, který si licencuje od společnosti ARM procesory přesně na míru, které budou zbaveny zbytečných částí, jako jsou WiFi a Bluetooth adaptéry a vybaveny většími cache pamětmi a výrazně více jádry, protože výkon čtyř jader je spolehlivě sražen potřebou komunikací. Nejlépe by již měly být postaveny na 64bitové architektuře AArch64.

Kapitola 8

Závěr

Cílem této práce bylo navrhnout a sestavit jednoduchý cluster složený z kitů Odroid XU4 postavených na nízkopříkonové architektuře ARM. Ten následně otestovat pomocí benchmarků měřících výkon a spotřebu a získané výsledky porovnat s již existujícími clustery, což se vše povedlo.

Cluster byl sestaven ze čtyř kitů propojených sítí Ethernet a externího NFS úložiště. Po prostudování existujících clusterů byly pro správu použity programy PBS Torque a EasyBuild. Výkon postaveného clusteru byl otestován hlavně pomocí benchmarků HPL pro výkon, STREAM pro propustnost paměti a OSU Micro-Benchmarks pro výkon rozhraní MPI.

Na jednom kitu se podařilo dosáhnout maximálního výkonu 8,5 GFLOP/s ve dvojité přesnosti, což byly při dané frekvenci asi 2/3 maximálního teoretického výkonu, což je velmi slušný výsledek. Při spojeném výkonu všech čtyř kitů dosahoval cluster v benchmarku HPL výkonu 24,5 GFLOP/s ve dvojité přesnosti, což je asi 10× menší výkon, než kterého dosahovaly dva spojené 8jádrové procesory Intel Xeon E5-2665. Reálná propustnost pamětí byla naměřena okolo 5,5 GB/s. Při testech spotřeby bylo změřeno, že cluster jako celek dosahuje efektivity výpočtu přibližně 0,58 GFLOP/s/W při spotřebě okolo 10 W. Hodnota efektivity byla podobná jako u testovaných Xeonů.

Architektura ARM má v oblasti HPC rozhodně potenciál. Pro další využití by ale mělo smysl přejít na generaci kitů s architekturou ARMv8, které jsou postaveny na 64bitové architektuře a to zejména kvůli podpoře vektorových instrukcí pro NEON ve dvojité přesnosti a problémům HPC softwaru na 32bitové architektuře ARM.

Zajímavé by také mohlo být použít kity postavené na čipech Nvidia Tegra a zjistit jak budou škálovat výpočty na grafických akcelerátorech na větším množství kitů.

Práce na projektu mi umožnila nahlédnout do světa mobilních procesorů ARM a přinesla mi cenné zkušenosti v oblasti stavby a správy superpočítačových clusterů.

Literatura

[1] Altair Engineering, Inc.: *Altair to Open Source PBS Professional® HPC Technology in 2016*. [online]. 2016 [cit. 2016-05-04].
URL http://www.altair.com/newsdetail.aspx?news_id=11199

[2] ARM Limited.: *NEON Support in Compilation Tools Development Article*. [online]. 2009 [cit. 2016-05-01].
URL http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dht0004a/ch01s04s03.html

[3] ARM Limited.: *NEON Support in Compilation Tools Development Article*. [online]. 2015 [cit. 2016-05-01].
URL http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.den0024a/CEGDJGGC.html

[4] ARM Ltd.: *Mali T628 - ARM*. [online]. 1995-2016 [cit. 2016-04-18].
URL http://www.arm.com/products/multimedia/mali-gpu/high-performance/mali-t628.php

[5] ARM Ltd.: *ARM NEON*. [online]. 2015 [cit. 2016-01-10].
URL http://www.arm.com/products/processors/technologies/neon.php

[6] ARM Ltd.: *ARM Processor Architecture*. [online]. 2015 [cit. 2016-01-10].
URL
http://www.arm.com/products/processors/instruction-set-architectures/

[7] CompuGreen, LLC.: *The Green500 List - November 2015*. [online]. 2007-2013 [cit. 2016-05-15].
URL http://www.green500.org/lists/green201511

[8] Elinux.org: *RPi Performance*. [online]. 2015 [cit. 2016-01-10].
URL http://www.roylongbottom.org.uk/linpack%20results.htm

[9] Fosdick, L. D.: *An Introduction to High-Performance Scientific Computing*, kapitola 11. Morgan Kaufmann Publishers, 1996, ISBN 0-262-06181-3, str. 354.

[10] Hardkernel co., Ltd.: *ODROID-XU4*. [online]. 2015 [cit. 2016-05-10].
URL http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825

[11] Intel Corporation: *Intel® Xeon® Processor E5-2665*. [online]. 2012 [cit. 2016-05-11].
URL http://ark.intel.com/products/64597/Intel-Xeon-Processor-E5-2665-20M-Cache-2_40-GHz-8_00-GTs-Intel-QPI

[12] Intel Corporation: *Intel® Xeon® Processor E5-2680 v3*. [online]. 2014 [cit. 2016-05-11].
URL http://ark.intel.com/products/81908/Intel-Xeon-Processor-E5-2680-v3-30M-Cache-2_50-GHz

[13] IT4Innovations: *Intel Performance Counter Monitor*. [online]. 2016 [cit. 2016-05-10].
URL https://docs.it4i.cz/anselm-cluster-documentation/software/debuggers/intel-performance-counter-monitor

[14] Jack J. Dongarra: *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*. [online]. 2016 [cit. 2016-01-08].
URL http://www.netlib.org/benchmark/hpl/

[15] Jack J. Dongarra, Piotr Luszczek, Antoine Petitet: *The LINPACK Benchmark: Past, Present, and Future*. [online]. 2001 [cit. 2016-01-08].
URL http://www.netlib.org/utk/people/JackDongarra/PAPERS/hpl.pdf

[16] Samsung Electronics Co., Ltd.: *Exynos 5 Octa (5422)*. [online]. 2014 [cit. 2016-04-18].
URL http://www.samsung.com/semiconductor/minisite/Exynos/w/solution.html#?v=octa_5422

[17] T. Deakin,S. McIntosh-Smith: *GPU-STREAM: Benchmarking the achievable memory bandwidth of Graphics Processing Units*. [online]. 2015 [cit. 2016-05-15].
URL http://sc15.supercomputing.org/sites/all/themes/SC15images/tech_poster/poster_files/post150s2-file3.pdf

[18] TOP500.org: *TOP500 Lists - November 2015*. [online]. 2015 [cit. 2016-01-10].
URL http://www.top500.org/lists/2015/11/

[19] Travis Lanier.: *Exploring the Design of the Cortex-A15 Processor*. [online]. 2011 [cit. 2016-05-13].
URL http://www.arm.com/files/pdf/AT-Exploring_the_Design_of_the_Cortex-A15.pdf

Přílohy

Seznam příloh

Příloha A

Výsledky benchmarku Linpack z testu přepínačů

Soubor A.1: Výsledky benchmarku Linpack v GFLOP/s v jednoduché přesnosti pro různé kombinace přepínačů.

```
#  -O3  -O2  -O1  -O0
01 1,13 1,12 1,16 0,26 -mtune=cortex-a15 -mfpu=neon-vfpv4
02 2,26 1,08 1,17 0,24 -mfpu=neon-vfpv4 -ffast-math
03 1,13 1,12 1,17 0,26 -mtune=cortex-a15 -mfpu=vfpv3-d16 -ffast-math -ftree-vectorize
04 2,28 1,12 1,17 0,26 -mtune=cortex-a15 -mfpu=neon-vfpv4 -ffast-math
05 1,13 1,12 1,17 0,26 -mtune=cortex-a15 -mfpu=vfpv3-d16 -ftree-vectorize
06 1,10 1,07 1,17 0,24 -mfpu=vfpv3-d16 -ffast-math
07 1,09 1,08 1,17 0,24 -mfpu=neon-vfpv4 -ftree-vectorize
08 1,10 1,08 1,17 0,24 -mfpu=vfpv4
09 1,13 1,12 1,17 0,26 -mtune=cortex-a15 -mfpu=vfpv4
10 1,13 1,12 1,17 0,26 -mtune=cortex-a15 -mfpu=vfpv3-d16
11 1,10 1,08 1,17 0,24 -mfpu=vfpv4 -ffast-math
12 1,13 1,12 1,17 0,26 -mtune=cortex-a15 -mfpu=vfpv4 -ffast-math -ftree-vectorize
13 1,13 1,12 1,17 0,26 -mtune=cortex-a15 -mfpu=vfpv3-d16 -ffast-math
14 1,13 1,12 1,17 0,26 -mtune=cortex-a15 -mfpu=vfpv4 -ffast-math
15 2,26 2,16 1,55 0,24 -mfpu=neon-vfpv4 -ffast-math -ftree-vectorize
16 1,10 1,08 1,17 0,24 -mfpu=neon-vfpv4
17 1,10 1,08 1,17 0,24 -mfpu=vfpv4 -ftree-vectorize
18 1,13 1,12 1,17 0,26 -mtune=cortex-a15 -mfpu=neon-vfpv4 -ftree-vectorize
19 2,28 2,20 1,57 0,26 -mtune=cortex-a15 -mfpu=neon-vfpv4 -ffast-math -ftree-vectorize
20 1,10 1,08 1,17 0,24 -mfpu=vfpv4 -ffast-math -ftree-vectorize
21 1,09 1,08 1,17 0,24 -mfpu=vfpv3-d16 -ftree-vectorize
22 1,10 1,08 1,16 0,24 -mfpu=vfpv3-d16 -ffast-math -ftree-vectorize
23 1,10 1,08 1,17 0,24 -mfpu=vfpv3-d16
24 1,13 1,12 1,17 0,26 -mtune=cortex-a15 -mfpu=vfpv4 -ftree-vectorize
```

Soubor A.2: Výsledky benchmarku Linpack v GFLOP/s ve dvojité přesnosti pro různé kombinace přepínačů.

```
#  -O3   -O2   -O1   -O0
01 1,020 1,003 0,979 0,209 -mfpu=vfpv3-d16 -ftree-vectorize
02 1,021 0,999 0,978 0,208 -mfpu=neon-vfpv4 -ffast-math -ftree-vectorize
03 1,021 1,010 0,983 0,244 -mtune=cortex-a15 -mfpu=vfpv4 -ftree-vectorize
04 1,020 1,011 0,984 0,243 -mtune=cortex-a15 -mfpu=vfpv3-d16 -ffast-math -ftree-vectorize
05 1,020 1,002 0,979 0,209 -mfpu=vfpv4 -ftree-vectorize
06 1,020 1,011 0,984 0,244 -mtune=cortex-a15 -mfpu=neon-vfpv4 -ffast-math -ftree-vectorize
07 1,020 1,013 0,986 0,243 -mtune=cortex-a15 -mfpu=vfpv4
08 1,021 1,002 0,977 0,209 -mfpu=neon-vfpv4 -ftree-vectorize
09 1,021 1,012 0,986 0,243 -mtune=cortex-a15 -mfpu=vfpv4 -ffast-math
10 1,019 0,998 0,979 0,208 -mfpu=vfpv3-d16 -ffast-math -ftree-vectorize
11 1,020 1,014 0,987 0,243 -mtune=cortex-a15 -mfpu=neon-vfpv4 -ffast-math
12 1,021 1,013 0,980 0,244 -mtune=cortex-a15 -mfpu=vfpv3-d16
13 1,021 1,004 0,980 0,209 -mfpu=vfpv4
14 1,021 1,001 0,980 0,208 -mfpu=neon-vfpv4 -ffast-math
15 1,021 1,012 0,987 0,244 -mtune=cortex-a15 -mfpu=vfpv3-d16 -ffast-math
16 1,020 1,000 0,979 0,208 -mfpu=vfpv4 -ffast-math -ftree-vectorize
17 1,021 1,001 0,980 0,208 -mfpu=vfpv3-d16 -ffast-math
18 1,021 1,009 0,984 0,243 -mtune=cortex-a15 -mfpu=vfpv3-d16 -ftree-vectorize
19 1,021 1,010 0,983 0,243 -mtune=cortex-a15 -mfpu=neon-vfpv4 -ftree-vectorize
20 1,020 1,005 0,980 0,208 -mfpu=vfpv3-d16
21 1,020 1,013 0,987 0,244 -mtune=cortex-a15 -mfpu=neon-vfpv4
22 1,021 1,000 0,980 0,208 -mfpu=vfpv4 -ffast-math
23 1,021 1,011 0,983 0,243 -mtune=cortex-a15 -mfpu=vfpv4 -ffast-math -ftree-vectorize
24 1,021 1,005 0,980 0,208 -mfpu=neon-vfpv4
```

Příloha B

Konfigurační soubor HPL.dat

Soubor B.1: Konfigurační soubor HPL.dat pro benchmark HPL, se kterým bylo dosaženo nejlepšího výsledku pro jeden kit – 8,5 GFLOP/s při frekvenci 1600MHz

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
12096        Ns
1            # of NBs
192          NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
2            Ps
2            Qs
16.0         threshold
1            # of panel fact
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
2            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
2            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
0            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```