Gottfried Wilhelm Leibniz Universität Hannover
Institut für Verteilte Systeme
Distributed Computing & Security Group

Seminarausarbeitung
im Studiengang Informatik (M.Sc.)

Schriftliche Ausarbeitung für Seminar Aspekte Verteilte Systeme

Verfasser: Zhang, Zijian
Betreuer: Dipl.-Inform. H. Tobaben
Datum: 3. Juli 2016

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Codeverzeichniss

Kapitel 1

HPC

1.1 Was ist HPC

HPC(Auf Englisch High Performance Computing), oder Rechnen mit dem Hilfer von Superrechner, ist typischweise Supercomputer mit großen Anzahl der Prozessoren, die auf gemeisame Peripheriegeräte und eine teiweise gemeisamen Hauptspeicher zugreifen können, die die auf dem Wikipedia beschrieben werden.

1.2 Warum HPC

Um die Lösung maßstabreicher Probleme zu finden. Diese Probleme hat dem Merkmal, dass sie auf viele homogenisierende Teilprobleme bestanden, zum Beispiel die auf dem Bereich Hydromechanik, Biologie oder atmosphärische Wissenschaften. Um das Rechnungsprozess zu beschleunigen, ist parallele Rechnung sehr hilfreich. Das ist auch der Grund dafür, warum Supercomputers werden von Institute alle Land immer noch untersucht.

1.3 HPC und Energiebedarf

Unter der gleichem Architechtur besitzt ein Supercomputer je mehr Prozessoren verbraucht es mehr Energie. Mit der Leistung von 93.000,00 TeraFLOPS beträgt der Energiebedarf von Sunway TaihuLight 15.370kW, was ist eigentlich zehr energieaufwändig. Aber dieser Zustand ist verbesserbar. Eine Seit durch den Fortschritten der Technik kann man die Architektur von Supercomputer monifizieren, um zum schluß die Energiebedarf per Prozessor zu senken, andere Seit ist die Softwareimpimentation auch verbesserbar. In diser Ausarbeitung biete ich ein paar Vergleichen an, der viele Energiebedarf-relevante Aspekte identifizieren.

Kapitel 2

Versuchen

2.1 Untersuchungsplattform

Unsere Untersuchung wurde durchgeführt auf dem Plattform wie unter:

Hardwaresystem	ODROID-XU3 Lab Environment(mit ARM Cortex-A7 1.4Ghz und Cortex-A15 2.0Ghz big.LITTLE architecture jeweils 4 kerne)
Betribssystem	Ubuntu 15.10 mit ssh Zugriff und shared storage durch NFS server
Test-Benchmark	NAS Parallel Benchmarks
Tasks	LU Dekomposition und Gauß'sche Elimination(LU)
	und konjugierender Gradient(CG)
Task-Maßstab	siehe Tablett 2.1
Implementationssprache	Java
Implementationsmethode	OpenMP und MPI(Message Passing Interface)
Messungsgeräte	ODROID Smart Power Device

Tabelle 2.1: Untersuchungsplattform

		Class A	Class B
CG	Size	14000	75000
	Iteration	15	75
LU	Size	64x64x64	102x102x102
	Iteration	250	250

Tabelle 2.2: Task-Maßstab

2.2 Energiebedarf-relevante Aspekte

2.2.1 Energiebedarf und Architektur

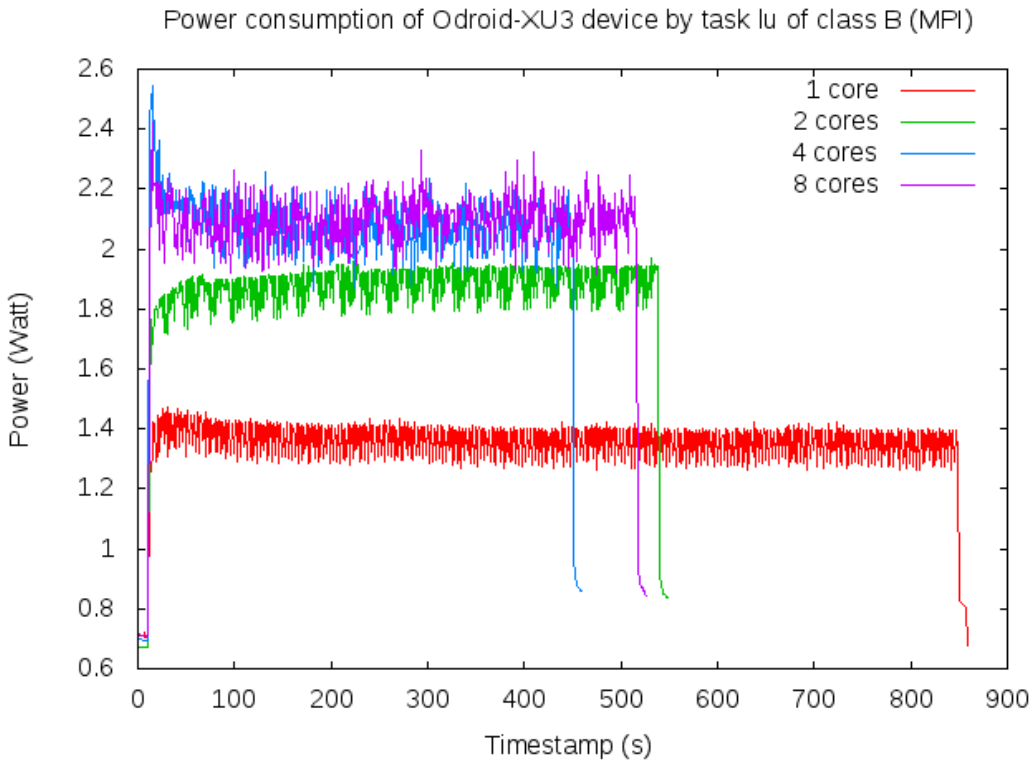

Abbildung 2.1: Leistung und Kernenutzung

Wie gezeigt in Abbildung 2.1 und sein Integral Abbildung 2.2. Die Rechnung befasst sich um das Task von LU und mit BWenn es nur ein Kern aktiviert ist, ist die Energiebedarf des Computersystems am höchsten und dauert die Rechung am längsten. Deshalb die Konfiguration mit nur ein aktivierende Kern ist auf jeden Fall nicht effizient.
Aber es ist auch komisch, dass die Zeit- und Energiebedarf des Task von 8 Kerne sind beide höher als die von 4 Kerne. Laut der Dokumentation von ODROID-XU3 kann man feststellen, dass es 4 Kerne von Cortex-A7 und 4 Kerne von Cortex-A15 auf der Tafel sich befindet. Wenn es nur 4 Kerne aktiviert ist, sind die 4 Kerne mit niedrigere ID, d.h. 4 Kerne von Cortex-A7 laufbar.
Der Grunden dafür kann man mit dem Hilfe Dokumentationen und Versuchergebnisse vermuten. Rechnungsdatenumtauschung innerhalb ein CPU ist immer noch effizienter als die über 2 CPUs. Darum ist die Zeitbedarf mit 4 Kerne nideriger. Beschrieben von Webseite von ARM und in Beachtung von Messungsergebnis in Abbildung 2.3, ist die Leistung des Cortex-A15 höher als die des Cortex-A7 und für die Energiebedarf vice versa. Deshalb für eine Rechnungsarbeit mit so moderatem

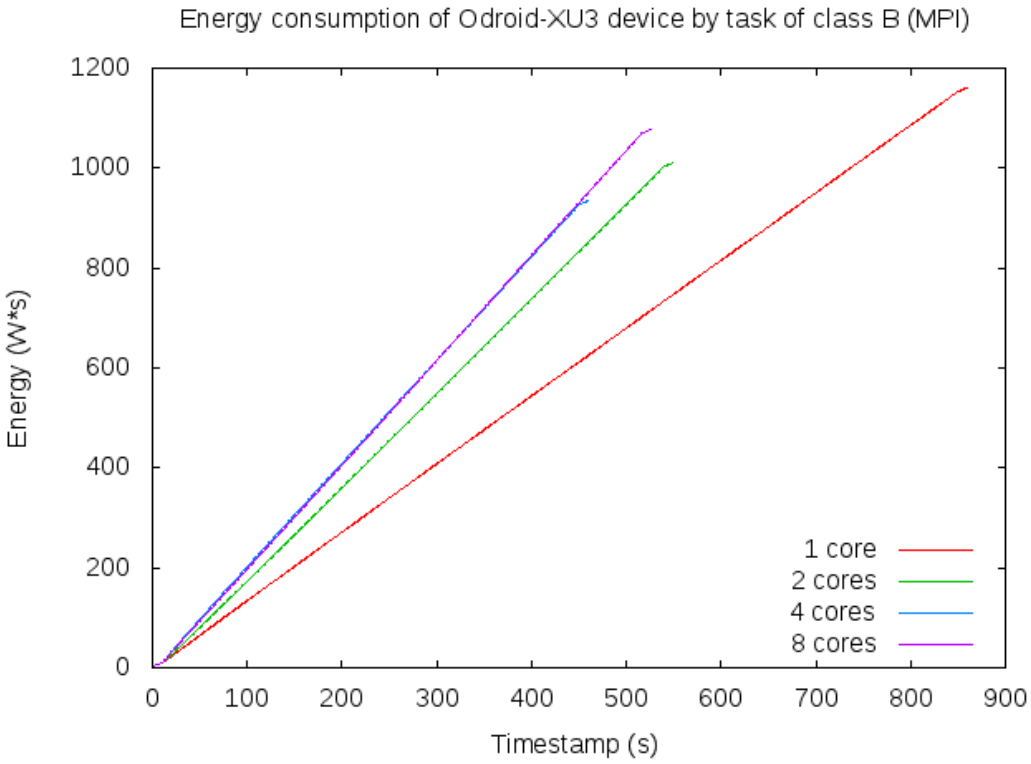

Abbildung 2.2: Energiebedarf und Kernenutzung

Maßtab soll man die energieeffiziente CPUs beforzugen.

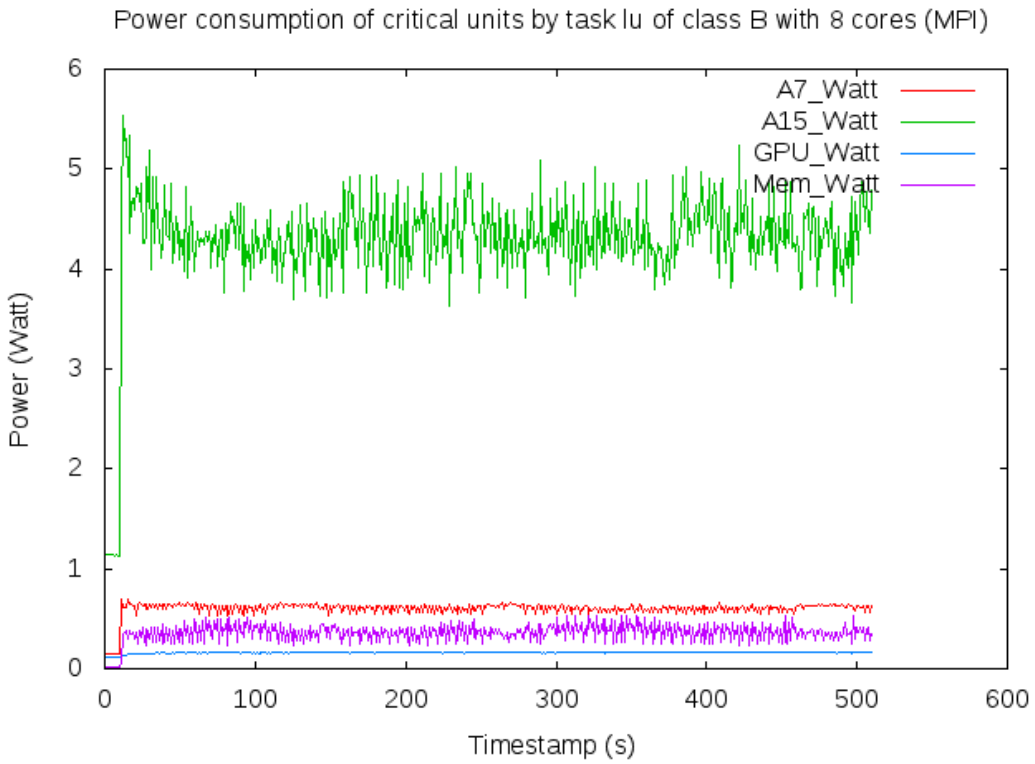

Abbildung 2.3: Leistung kritischer Teilen

Kapitel 3

Codes

In disem Kapitel werden alle automatisierte Skripte (zu kompilieren und messen) dargestellt

```
import os

result_dir = "~/res"
base_dir = "~/LAB/NAS/"
ver_dir= "NPB3.3.1/"
ver_short = "NPB3.3"
impls= ["MPI", "OMP"]

benchmark_names = ["CG", "LU"]
class_names = ["S", "W", "A", "B", "C", "D"]

for impl in impls:
     cp_command = "cp "+base_dir+impl+"/make.def "+
                          base_dir+ver_dir+ver_short+
                               "-"+impl+"/config/"

     os.system(cp_command)

     for bn in benchmark_names:
         for cn in class_names:
             for core_num in [1,2,4,8]:
                 make_command = "cd "+base_dir+ver_dir+
                                ver_short+"-"+impl+
                                "/;make "+bn

                 if impl == "MPI":
```

```
                    make_command+=" NPROCS="+
                                    str(core_num)

                elif core_num>1:
                    break

                make_command += " CLASS="+cn
                os.system(make_command)
```

Listing 3.1: compile.py

```
#!/bin/bash

res_dir="/home/user5/res/"
if [ ! -f $resultdir ];
then
 mkdir -p $resultdir
fi

basedir="/home/user5/LAB/NAS/"
verdir="NPB3.3.1/"
vershort="NPB3.3"

impls="MPI OMP"
benchmark_names="cg lu"
class_names="A B"

core_nums="1 2 4 8"

mes_sensor="stdbuf -oL read-xu3-sensors -c -t"
mes_power="stdbuf -oL smartpower -c -t"

for impl in $impls;
do
 for bn in $benchmark_names;
 do
  for cn in $class_names;
  do
   for core_num in $core_nums;
   do
    res_folder=$res_dir"/"$impl"/"
```

```
if [ ! -f $res_folder ];
then
 mkdir -p $res_folder
fi

sensor_res_filename=$res_folder$bn"."\
                    $cn"."$core_num\
                    ".sensor.txt"
power_res_filename=$res_folder$bn"."\
                   $cn"."$core_num\
                   ".power.txt"

if [ -e $sensor_res_filename ] &&
   [ `wc -l $sensor_res_filename |
     cut -f 1 --delimiter=" "` != "0" ] &&
   [ -e $power_res_filename ] &&
   [ `wc -l $power_res_filename |
     cut -f 1 --delimiter=" "` != "0" ];
then
 continue
fi

($mes_sensor)>$sensor_res_filename&
pid_mes_sensor=$!
($mes_power)>$power_res_filename&
pid_mes_power=$!

sleep 10

cd "/home/user5/LAB/NAS/NPB3.3.1/NPB3.3-"$impl"/bin"
if [ $impl == "MPI" ];
then
 run_cmd="mpirun.mpich -np $core_num ./"\
         $bn"."$cn"."$core_num
else
 export OMP_NUM_THREADS=$core_num
 run_cmd="./"$bn"."$cn".x"
fi
```

```
    ($run_cmd)
    sleep 10

    kill $pid_mes_sensor
    kill $pid_mes_power

    sync
    sleep 2
   done
  done
done
done
```

Listing 3.2: meassure.sh