Tools for Computational Physics
Week 3, Lecture 2: Optimization:
Picking the right tools for the job

R. Rousseau
SISSA, 2-4 Via Beirut,
Trieste, Italy
Outline

• Introduction
• Review: Memory Hierarchy
• Classifying Codes
• Optimizing performance for:
  – I/O intense codes
  – Memory Intense codes
  – I/O and memory intense codes
  – CPU intense codes.
• Conclusion
Introduction

The goal of this lecture is to provide suggestions on how to optimize for performance different types of computational tasks.

Fundamental to this advocated approach is an understanding of what are the computational needs of a given type of problem and how this is limited by the size and speed of the various computer components.

A hierarchy of problem types will be introduced and some rough guidelines on where to spend time, energy and effort to obtain maximum use of (limited) resources will be presented.
The Memory Hierarchy: A Review

- CPU
- Data
- Instructions
- Addresses
- RAM
- VIRTUAL

CPU Speed vs. Size

- Processor side
- System side

Speed:
- ns
- μs
- ms-s

Size:
- Element
- Line
- Page
Where to focus your time and Energy

Define the *physics* of your problem and your computational needs

Using the memory Hierarchy determine where the computational Bottlenecks will be for your problem.

If possible, try to design your software to be more reliant on the faster parts of the computer: i.e. minimize I/O

If possible try to alter or procure hardware (operating system configuration) that will maximize performance on the weakest part of you code.
## Computational Needs of Different Problems

<table>
<thead>
<tr>
<th>Computational Requirement</th>
<th>Examples Physics</th>
<th>Example Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Intense</td>
<td>Classical Molecular Dynamics, Monte Carlo</td>
<td>GROMACS</td>
</tr>
<tr>
<td>Memory Intense</td>
<td>Planewave electronic structure</td>
<td>CPMD. PWSCF</td>
</tr>
<tr>
<td>I/O Intense</td>
<td>Quantum Chemistry, Linear Response solid state, Climate Modeling, Statistical</td>
<td>Gaussian, Games</td>
</tr>
<tr>
<td></td>
<td>Analysis on Large data sets</td>
<td>PWSCF</td>
</tr>
<tr>
<td>I/O and Memory</td>
<td>QMC, FLAPW</td>
<td></td>
</tr>
</tbody>
</table>
What to do if your bottleneck is I/O

Panic, do something unhealthy and unconstructive
Until you can reduce I/O all the nice
coding tricks won't help much

Can you reduce I/O?
Scratch files of ca 5-10GB could be put in
memory for parallel jobs or on 64bit machines
NEVER WRITE FORMATTED ASCII SCRATCH USE
UNFORMATTED BINARY

If not, is there a hardware solution?
Here mainframes are a big plus.
On a cluster always run on local scratch

Build, buy or install a RAID array
Redundant Array of Inexpensive Disks (RAID)

One can improve I/O performances N-fold by using N disks together.

- **Mirroring RAID (hardware)**
  - hda1
  - hda2
  - hda3

- **Striping (R)AID (software)**
  - hda1
  - hda2
  - hda3

*Three disk heads are better than one!*
What is the difference, Which is better

Hardware RAID:
- A machine with a lot of disks
- cheaper with lots of mother boards supporting it
- REDUNDANT i.e. if one disk dies you still have copies of your data
- useful to store critical data.

Software RAID
- Does not need special hardware.
- IS NOT REDUNDANT: one disk dies you loose.
- stable in Linux 2.4 or later
- for block data read/writes as used in Fortran it is faster than hardware
- cheap all modern Linux distributions support it (relatively easy to set up)
- cpu overhead.
- easy to configure: create a raitab file and turn on the raid kernel module

Suggestions for good health:
Disks need power so make sure if you have a good power supply
Do not put RAID disks on IDE slave controller (kills performance)
buy external disk controller if you have to.
“Not doin’ too bad, Not doin’ too bad at all”

- Character read write is very slow and cpu intensive
- RAID 0 is ca 1.7 times faster then direct rw
- NFS (one user only) is reasonable.
- RAID 1 and 5 not so great.

<table>
<thead>
<tr>
<th>Machine</th>
<th>Size</th>
<th>K/sec</th>
<th>%CP</th>
<th>K/sec</th>
<th>%CP</th>
<th>K/sec</th>
<th>%CP</th>
<th>K/sec</th>
<th>%CP</th>
<th>K/sec</th>
<th>%CP</th>
<th>/sec</th>
<th>%CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>opt13</td>
<td>8G</td>
<td>40083</td>
<td>98</td>
<td>50569</td>
<td>20</td>
<td>20888</td>
<td>5</td>
<td>30423</td>
<td>67</td>
<td>62816</td>
<td>7</td>
<td>176.3</td>
<td>0</td>
</tr>
<tr>
<td>opt00-r1</td>
<td>8G</td>
<td>26943</td>
<td>65</td>
<td>42824</td>
<td>18</td>
<td>19014</td>
<td>5</td>
<td>25880</td>
<td>59</td>
<td>48473</td>
<td>6</td>
<td>246.7</td>
<td>1</td>
</tr>
<tr>
<td>opt00-r5</td>
<td>8G</td>
<td>28233</td>
<td>68</td>
<td>42130</td>
<td>18</td>
<td>16324</td>
<td>5</td>
<td>19824</td>
<td>45</td>
<td>30382</td>
<td>4</td>
<td>288.5</td>
<td>1</td>
</tr>
<tr>
<td>magicslim-r0</td>
<td>2G</td>
<td>25013</td>
<td>92</td>
<td>82036</td>
<td>15</td>
<td>36152</td>
<td>7</td>
<td>22586</td>
<td>89</td>
<td>95345</td>
<td>13</td>
<td>262.7</td>
<td>0</td>
</tr>
<tr>
<td>magicslim</td>
<td>2G</td>
<td>25149</td>
<td>90</td>
<td>54658</td>
<td>9</td>
<td>22964</td>
<td>4</td>
<td>23010</td>
<td>86</td>
<td>57687</td>
<td>7</td>
<td>178.7</td>
<td>0</td>
</tr>
<tr>
<td>pickett-r0</td>
<td>2G</td>
<td>17304</td>
<td>81</td>
<td>33686</td>
<td>13</td>
<td>16545</td>
<td>7</td>
<td>17480</td>
<td>79</td>
<td>77944</td>
<td>23</td>
<td>198.5</td>
<td>0</td>
</tr>
<tr>
<td>bobmarley</td>
<td>2G</td>
<td>18303</td>
<td>95</td>
<td>25831</td>
<td>15</td>
<td>14984</td>
<td>6</td>
<td>19155</td>
<td>91</td>
<td>83915</td>
<td>20</td>
<td>212.7</td>
<td>0</td>
</tr>
<tr>
<td>hampton</td>
<td>2G</td>
<td>15970</td>
<td>76</td>
<td>24071</td>
<td>8</td>
<td>8853</td>
<td>2</td>
<td>18525</td>
<td>73</td>
<td>40479</td>
<td>6</td>
<td>111.1</td>
<td>0</td>
</tr>
<tr>
<td>schubert</td>
<td>2G</td>
<td>16823</td>
<td>97</td>
<td>26595</td>
<td>16</td>
<td>9042</td>
<td>4</td>
<td>15394</td>
<td>75</td>
<td>28456</td>
<td>8</td>
<td>109.7</td>
<td>0</td>
</tr>
<tr>
<td>ravel-r5</td>
<td>3G</td>
<td>16139</td>
<td>90</td>
<td>42481</td>
<td>34</td>
<td>18987</td>
<td>11</td>
<td>15030</td>
<td>74</td>
<td>51327</td>
<td>16</td>
<td>307.7</td>
<td>1</td>
</tr>
<tr>
<td>ravel-hd</td>
<td>3G</td>
<td>16400</td>
<td>87</td>
<td>18061</td>
<td>10</td>
<td>5480</td>
<td>2</td>
<td>15006</td>
<td>76</td>
<td>29659</td>
<td>6</td>
<td>117.1</td>
<td>0</td>
</tr>
<tr>
<td>magicslim-1000Mb</td>
<td>2G</td>
<td>19449</td>
<td>88</td>
<td>42302</td>
<td>10</td>
<td>25992</td>
<td>17</td>
<td>19492</td>
<td>87</td>
<td>60765</td>
<td>12</td>
<td>3742.3</td>
<td>34</td>
</tr>
<tr>
<td>bobmarley-100Mb</td>
<td>2G</td>
<td>10901</td>
<td>59</td>
<td>10927</td>
<td>3</td>
<td>4879</td>
<td>3</td>
<td>11773</td>
<td>63</td>
<td>14666</td>
<td>4</td>
<td>238.0</td>
<td>2</td>
</tr>
</tbody>
</table>

machine description:
- opt13  (dual 2GHz opteron with single seagate IDE HD, 4GB RAM)
- opt00-r1 (dual 2GHz opteron with hardware ide2scsi raid-1, 4GB RAM)
- opt00-r5 (dual 2GHz opteron with hardware ide2scsi raid-5, 4GB RAM)
- magicslim-r0 (2GHz athlon 64 3200+, dual IDE software raid-0, 1GB RAM)
- magicslim (2GHz athlon 64 3200+, single IDE HD same as in raid-0, 1GB RAM)
**The Bottom line: code speed up**

With RAID 0 there is a small CPU time cost but Can get 1.6-2.0 times speed up relative to SCSI or IDE. Note even old hardware with RAID 0 can be better than New hardware without RAID 0.

### Conventional SCF Quantum Chemistry Program

16 SCF Iterations using the Integral Files:

<table>
<thead>
<tr>
<th>Machine</th>
<th>Disk</th>
<th>Cputime</th>
<th>Wall Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Athlon64 2.0GHz</td>
<td>SCSI 10k rpm</td>
<td>58.5 min</td>
<td>160.0 min</td>
</tr>
<tr>
<td>Athlon64 2.0GHz</td>
<td>IDE 7.2k rpm</td>
<td>58.5 min</td>
<td>128.0 min</td>
</tr>
<tr>
<td>Athlon64 2.0GHz</td>
<td>IDE RAID-0</td>
<td>60.5 min</td>
<td>77.0 min</td>
</tr>
<tr>
<td>AthlonXP 1.53GHz</td>
<td>IDE RAID-0 (old)</td>
<td>114 min</td>
<td>148 min</td>
</tr>
<tr>
<td>Athlon 650MHz z</td>
<td>IDE RAID-0</td>
<td>249 min</td>
<td>266 min</td>
</tr>
</tbody>
</table>

*Integral files are ca 18GB*
Memory Bottlenecks

Welcome to a nicer world: your life is easier
Find out what routines take the most CPU time
and focus your energy there*

Area to focus to improve code:
• Make optimal use of math libs.
• Matrix/Matrix to replace Matrix vector
• Follow hints from previous lecture
• Try vectorization by compiler.
• Avoid a lot of nested if/then

Hardware
• Disk is not a problem go for 2\textsuperscript{nd}/3\textsuperscript{rd} fastest ID
• Optimize hardware for best memory performance
• Better CPUs will be needed if your code is not fully optimized
  (64 bit: alpha, opteron, power PC, Itanic)

If you use one type of platform predominantly
then optimize your code for that platform

*If possible use a profiler like gprof (compile with flag −p −g (intel) −pg (gnu,pgi)
Here is a typical messy make file where several, math libs, compilers Compiler options etc are being tried.
Modifying key subroutines for the

Since the code spends a huge amount of time within the fft routines
A good place to improve performances is to look for
machine specific variables within the fft subroutines.
CPMD: Sample timings

Sample job Au\textsubscript{72}Au’\textsubscript{72}(SCH\textsubscript{3})\textsubscript{8} ca 1000 electrons
Basis 20 and 40 Ry plane wave cutoff.
Timing for a single step of CP dynamics

<table>
<thead>
<tr>
<th>Mem freq</th>
<th>CPU TIME/step</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1cpu</td>
</tr>
<tr>
<td>266</td>
<td>421</td>
</tr>
<tr>
<td>333</td>
<td>381</td>
</tr>
<tr>
<td>400</td>
<td>358</td>
</tr>
</tbody>
</table>

1.5GHz Itanium 2 Intel 8.0 MKL: 277
Opteron after code optimization: 316

Note Itanium price \approx 2X opteron price

Moral of the story brains size beats budget size (but a big budget isn’t a bad thing)
## Steps toward a faster code

<table>
<thead>
<tr>
<th></th>
<th>Compiler</th>
<th>Library</th>
<th>Fix</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 Ry Opteron 2.0GHz, 1MB cache, 400Mhz Memory</td>
<td>PGI 5.1</td>
<td>Atlas</td>
<td>-</td>
<td>1124</td>
</tr>
<tr>
<td></td>
<td>PGI 5.1</td>
<td>Atlas</td>
<td>fft tune</td>
<td>1049</td>
</tr>
<tr>
<td></td>
<td>PGI 5.2</td>
<td>ACML</td>
<td>fft tune</td>
<td>1024</td>
</tr>
<tr>
<td></td>
<td>IFC</td>
<td>Atlas/ACML</td>
<td>fft tune</td>
<td>990</td>
</tr>
</tbody>
</table>

By tuning the FFT routine to the platform a 7%-10% speed up was obtainable. By playing with math libraries this was further increased to 10-20% (depending on the job) by compiler options and math library improvements.

Note this is not huge but the routines of interest in CPMD were well optimized to begin with and relatively speaking there was not much room to play (the numbers are significantly better in worse -darker less frequented and forgotten-parts of the code).
**I/O and memory both needed.**

If possible try to get as much of your code into memory.
Minimize I/O

---

**Area to focus to improve code:**

- Make optimal use of math libs.
- Matrix/Matrix to replace Matrix vector
- Follow hints from previous lecture
- Try vectorization by compiler.
- Avoid a lot of nested *if/then*

---

**Hardware solutions:**
Mainframe is a good idea.
Normally you need few CPUS (not intrinsically parallel) but with good vectorization options and big cache: Itanic, opteron, Mac.
Fast local hard drive (RAID 0)
What machine is your best bet.

Small 4-way machine
Higher end server class
64 bit CPU
lots of registers (maybe cache).
Least optimized code runs better here than on PIVs
Lots of memory ca 16-32GB
Motherboards usually support hardware RAID
But you can put in lots of disks and also have a software RAID.
Down side usually more than 2X the price of dual machines of the same cpu.
What happens if my code depends only on CPU speed.

You are in Nirvana; ie a fabled land where the big (clock speed) = good (performance)

Software Solution: go parallel
The best way to work is to know what the software you wish to use or write does and know what part of computer architecture will be the weakest link.

Always try to optimize software you are writing (and plan to use extensively) and minimize I/O when possible.

If you can adapt the machine to the software you have an easy route to get an increase in performance for minimal work.

Another way to increase performance is to run jobs on more than 1 CPU Using parallel message passing libraries. This will be the subject of the next lecture.