Performance Tuning on Itasca

Shuxia Zhangh and Andrew Gustafson
Nov. 27, 2012
Outline

- What computer resources are available under Itasca umbrella?
- Has your code run efficient?
  - Profiling applications using VTUNE
    - Serial, OpenMP and MPI codes
    - Great fun to explore the flow of your code through the hardware hierarchies
- Can you produce more for the same amount of Service Units?
  - Parallelize and optimize your code
  - Run the job on the right queues
    - Memory needs and I/O management
- How to reduce the queuing time?
Itasca Umbrella

- "Nehalem EP" nodes
  1091 nodes, each having 8 cores
  sharing 24 GiB of system memory,
  Xeon X5560 "Nehalem EP" processors
- Sandybridge Nodes (blades)
  51 Sandy-Bridge nodes.
  each contains 2 eight-core E5-2670 processor chips (2.6 GHz)
  35 nodes with 64GB of memory
  8 nodes with 128GB of memory
  8 nodes with 256GB of memory
- Jay - HP ProLiant ML370 G6 server with
  a pair of 2.66 GHz Xeon X5550 quad-core chips
  64 GB of RAM
Itasca Umbrella

All the compute nodes
  • Same home directory
  • 40-gigabit QDR InfiniBand (IB) interconnect.
  • /lustre file system
  • Each node has its own local /scratch disk
    • Jay - 71 TB storage mounted as /scratch.sas
    • Nehalem nodes – 90 GB attached /scratch
    • SB nodes – 560 GB attached /scratch

Each of the sub systems
  • serves for a unique purpose
  • has special queue set to meet the needs
### Configuration of Itasca

<table>
<thead>
<tr>
<th>Queue (-q)</th>
<th>Wall clock -l walltime</th>
<th>Nodes -l nodes</th>
<th>Cores ppn</th>
<th>memory (mb) pmem</th>
<th>Service Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>devel</td>
<td>2:00:00</td>
<td>64</td>
<td>8</td>
<td>2500</td>
<td>debugging and development</td>
</tr>
<tr>
<td>long</td>
<td>48:00:00</td>
<td>17</td>
<td>8</td>
<td>2500</td>
<td>jobs need &gt; 24 hours</td>
</tr>
<tr>
<td>batch</td>
<td>24:00:00</td>
<td>1083</td>
<td>8</td>
<td>2500</td>
<td>parallel jobs SMP and MPI</td>
</tr>
<tr>
<td>oc</td>
<td>24:00:00</td>
<td>4</td>
<td>varied</td>
<td>2500</td>
<td>Serial jobs</td>
</tr>
<tr>
<td>jay</td>
<td>24:00:00</td>
<td>1</td>
<td>varied</td>
<td>7500</td>
<td>Visualization</td>
</tr>
<tr>
<td>sb</td>
<td>24:00:00</td>
<td>35</td>
<td>varied</td>
<td>7500</td>
<td>large memory</td>
</tr>
<tr>
<td>sb128</td>
<td>96:00:00</td>
<td>8</td>
<td>varied</td>
<td>15000</td>
<td>in between</td>
</tr>
<tr>
<td>sb256</td>
<td>96:00:00</td>
<td>8</td>
<td>varied</td>
<td>30000</td>
<td>extremely large memory</td>
</tr>
</tbody>
</table>
Use of VTune

VTune is a utility for profiling applications

- Both serial and parallel (OpenMP and MPI)
- Supported languages: FORTRAN, C and C++
- Equipped with both graphic user interface (GUI) and command-line script

Run VTune GUI

```
module load vtune
amplxe-gui
```

Run VTune command-line script

```
module load vtune
amplxe-cl
```
Vtune command-line interface

Serial and OpenMP jobs

```
module load vtune
amplxe-cl ./a.out
```

**Default:** no data collection unless `-collect` is specified.

To view which analysis types are available, run:

```
$ amplxe-cl -help collect
```

To get detailed information on the analysis type, run:

```
$ amplxe-cl -help collect ${analysis_type}
```

**Command Line Interface Reference:**

```bash
$ amplxe-cl -help collect ${analysis_type}
```

<table>
<thead>
<tr>
<th>${analysis-type}</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>concurrency</td>
<td>Concurrency</td>
</tr>
<tr>
<td>frequency</td>
<td>CPU Frequency</td>
</tr>
<tr>
<td>hotspots</td>
<td>Hotspots</td>
</tr>
<tr>
<td>kncbandwidth</td>
<td>Knights Corner Platform Bandwidth</td>
</tr>
<tr>
<td>lightweighthotspots</td>
<td>Lightweight Hotspots</td>
</tr>
<tr>
<td>locksandwaits</td>
<td>Locks and Waits</td>
</tr>
<tr>
<td>snb-access-contention</td>
<td>Access Contention - Sandy Bridge</td>
</tr>
<tr>
<td>snb-bandwidth</td>
<td>Bandwidth - Sandy Bridge</td>
</tr>
<tr>
<td>snb-branch-analysis</td>
<td>Branch Analysis - Sandy Bridge snb-client</td>
</tr>
<tr>
<td>snb-core-port-saturation</td>
<td>Core Port Saturation - Sandy Bridge</td>
</tr>
<tr>
<td>snb-cycles-uops</td>
<td>Cycles and uOps - Sandy Bridge</td>
</tr>
<tr>
<td>snb-general-exploration</td>
<td>General Exploration - Sandy Bridge</td>
</tr>
<tr>
<td>snb-loop-analysis</td>
<td>Loop Analysis - Sandy Bridge</td>
</tr>
<tr>
<td>snb-memory-access</td>
<td>Memory Access - Sandy Bridge</td>
</tr>
<tr>
<td>snb-port-saturation</td>
<td>Port Saturation - Sandy Bridge</td>
</tr>
</tbody>
</table>
Procedure of profiling applications

1. Compile the code
   \texttt{icc -O2 -openmp your.c -o your.exe}
2. Set the environment parameters and analysis type
   \texttt{export OMP_NUM_THREADS=4}
   \texttt{report_ID=r1}
   \texttt{analysis_type=hotspots}
3. Profile your application
   \texttt{amplxe-cl -collect \${analysis_type} -r \${report_ID} ./your.exe}
4. View and Analyze the reported results via GUI
   \texttt{amplxe-gui -r \${report_ID}}
5. Repeat the procedures of 2 to 4 and compare results.

Procedure of profiling MPI applications

1. Compile the code with intel MPI
   module load intel ompi/intel
   mpicc –O2 your.c –o your.exe

2. Set the environment parameters and analysis type
   report=MPI
   type=hotspots

3. Profile your application
   mpirun -n 8 amplxe-cl –c ${type} –r ${report} ./your.exe

4. View and analyze profiling results via GUI
   amplxe-gui –r ${report}

5. Repeat the procedures of 2 to 4, comparing results

Can you produce more for the given amount of SUs?

Yes, through code optimization and parallelization.

Run the job on the right hardware

A real-world application: Tsunami simulation
Depth-averaged 2D shallow water equations:

\[ \frac{\partial h}{\partial t} + \frac{\partial}{\partial x} (hu) + \frac{\partial}{\partial y} (hv) = 0 \]

\[ \frac{\partial}{\partial t} (hu) + \frac{\partial}{\partial x} (hu^2 + \frac{1}{2} gh^2) + \frac{\partial}{\partial y} (huv) = -gh \frac{\partial b}{\partial x} + S_{fx} \]

\[ \frac{\partial}{\partial t} (hv) + \frac{\partial}{\partial x} (huv) + \frac{\partial}{\partial y} (hv^2 + \frac{1}{2} gh^2) = -gh \frac{\partial b}{\partial y} + S_{fy} \]

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Physical meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>Depth of water</td>
</tr>
<tr>
<td>b</td>
<td>Depth of sea floor</td>
</tr>
<tr>
<td>u,v</td>
<td>Depth-averaged velocities</td>
</tr>
<tr>
<td>Sfx, Sfy</td>
<td>Friction forces</td>
</tr>
<tr>
<td>n</td>
<td>the Manning coefficient</td>
</tr>
</tbody>
</table>
Model setup

Impulsive disturbance - focal model
by Chen Ji et al at UCSB

Bathymetry: One minute resolution from
database of National Geophysical DATA

One-level grid

Serial run – too slow!!!
Use of Vtune to find **Hotspot**

- To launch Vtune
  module load vtune amplxe-gui
- create a new project
- launch application
- run hotspot analysis
- interpret results

**Hotsport Analysis**
Amdahl's Law

Potential speedup
= \frac{98.5}{98.5-89.2}
= 10.59
if step2 is fully parallelized

Potential speedup
= \frac{98.5}{98.5-78.5}
= 4.99
if flux2 is fully parallelized
Parallelization of STEP2 subroutine with OpenMP
loop 50 (do j = 0, my+i); loop 100 (do i = 0, mx+1)
100% of subroutine flux2 can run in parallel
Achieved over 75% speed up
A real-world application

Using KMP_AFFINITY for controlling thread affinity

<table>
<thead>
<tr>
<th>Package 0</th>
<th>Package 1</th>
<th>Package 2</th>
<th>Package 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 0 OS proc 0</td>
<td>Core 0 OS proc 1</td>
<td>Core 0 OS proc 2</td>
<td>Core 0 OS proc 3</td>
</tr>
<tr>
<td>Core 1 OS proc 12</td>
<td>Core 1 OS proc 13</td>
<td>Core 1 OS proc 14</td>
<td>Core 1 OS proc 15</td>
</tr>
<tr>
<td>Core 2 OS proc 4</td>
<td>Core 2 OS proc 5</td>
<td>Core 2 OS proc 6</td>
<td>Core 2 OS proc 7</td>
</tr>
<tr>
<td>Core 3 OS proc 16</td>
<td>Core 3 OS proc 17</td>
<td>Core 3 OS proc 18</td>
<td>Core 3 OS proc 19</td>
</tr>
<tr>
<td>Core 4 OS proc 8</td>
<td>Core 4 OS proc 9</td>
<td>Core 4 OS proc 10</td>
<td>Core 4 OS proc 11</td>
</tr>
<tr>
<td>Core 5 OS proc 20</td>
<td>Core 5 OS proc 21</td>
<td>Core 5 OS proc 22</td>
<td>Core 5 OS proc 23</td>
</tr>
</tbody>
</table>

The default map of Operation System process on physical cores (CL0)

Performance tuning of UMSI_GeoClaw
A real-world application with GeoClaw code

Performance tuning of UMSI_GeoClaw

Affinity combined with First Touch

Data initialization

4 threads job
Parallelization in the routines in the directory of armclaw related to the adaptive mesh refinement.

Some of the routines in geoclaw directory are also parallelized.

Dr. George gave us an unreleased OpenMP version

But subroutine step2 is sequential. It consumes 60% of time in the sequential run.

Parallelization of step2 should improve the performance
1-thread run with WUA_Clawn
4-thread run with WUA_Claw
### Call Stack

<table>
<thead>
<tr>
<th>Call Stack</th>
<th>CPU Time</th>
<th>CPU Time:Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>▼ ▼ Total</td>
<td>0s</td>
<td>100.0%</td>
</tr>
<tr>
<td>▼ ▼ clone</td>
<td>0s</td>
<td>82.5%</td>
</tr>
<tr>
<td>▼ ▼ main</td>
<td>0s</td>
<td>14.3%</td>
</tr>
<tr>
<td>▼ ▼ ▼ MAIN__</td>
<td>0s</td>
<td>14.3%</td>
</tr>
<tr>
<td>▼ ▼ tick__</td>
<td>0s</td>
<td>13.1%</td>
</tr>
<tr>
<td>▼ ▼ advanc__</td>
<td>0.088s</td>
<td>12.1%</td>
</tr>
<tr>
<td>▼ ▼ advanc__</td>
<td>0.234s</td>
<td>12.0%</td>
</tr>
<tr>
<td>▼ ▼ ▼ stepgrid__</td>
<td>3.523s</td>
<td>10.7%</td>
</tr>
<tr>
<td>▼ ▼ ▼ ▼ step2__</td>
<td>10.266s</td>
<td>6.1%</td>
</tr>
<tr>
<td>▼ ▼ ▼ step2__</td>
<td>13.674s</td>
<td>5.7%</td>
</tr>
<tr>
<td>▼ ▼ flux2__</td>
<td>18.390s</td>
<td>5.3%</td>
</tr>
<tr>
<td>▼ ▼ b4step2__</td>
<td>0.508s</td>
<td>4.3%</td>
</tr>
<tr>
<td>▼ ▼ src2__</td>
<td>5.821s</td>
<td>0.2%</td>
</tr>
</tbody>
</table>

8-thread run with WUA_Clawn

---

Supercomputing Institute
for Advanced Computational Research

[Logo of University of Minnesota: Driven to Discover]
Merging of UMSI_GeoClaw with UWA_Geoclaw

- Nested Parallelization
- First level parallelization - Adaptive mesh refinement
- Second-level parallelization – GeoClaw for wave sweeping in Step2
- Three essential run-time variables with Intel compiler
  - OMP_NUM_THREADS – AMR
  - NUM_THREADS – Step2
  - OMP_NESTED – to enable the nesting in run-time
UMSI and UWA merged: 8 threads for AMR, and 2 nested threads for step2 routine, hypo-threading enabled. *Almost 100% speed up.*
Inundation simulation of Tsunami Waves from the Tohoku 2011 Earthquake

Focal mechanism model by Guangfu Shao, Xiangyu Li, Chen Ji, and Takahiro Maeda.

One minute Bathymetry Data

BC: Open boundary specified on all ocean boundaries

Grid resolution: 1200m to 20m with 4-levels of AMR
Inundation simulation of Tsunami Waves from the Tohoku 2011 Earthquake

Comparison of simulated wave height and arrival time with the observed ones at the Buoy station 21418
Wave propagation at 0.5 hours after the impulsion
Better use of the SB nodes
Code Compiling options

Intel/12 compiler – default on itasca
  C++ : icpc
  C   : icc
  FORTRAN : ifort

Recommended for most codes

GNU compiler – v 4.4 or newer
  C++ : g++
  C   : gcc
  FORTRAN : gfortran

Only for the codes that cannot work with intel compiler
GNU compiler flags applicable or SB processor

-mavx
This enables the AVX instruction set

-march=corei7-avx
This specifies that the executable will be run on an AVX enabled current generation chip

-mtune=corei7-avx
This specifies that the executable should be speed tuned for a current generation AVX chip
Code Compiling options – Intel compiler

Key functionality – Advanced vector extension (AVX)

Recompiled your codes:

Adding flag **-xavx** to compiling options
Executable will run only on SB nodes. It will NOT run on other Itasca nodes.

or

Adding flag **-axavx** to compiling options
Executable will run on both SB and other Itasca nodes, but with AVE-SSE transition penalty on performance
Code Compiling options for better performance

Other common used options -O3 –ipo, PGO and –vec-report

- **O3** enables more aggressive optimizations
- **ipo** enables the optimizations between procedures in separate source files.

**PGO** Profile-guided optimization is an iterative process

**Note:** Correct results must be obtained
Profile-Guided Optimization

1. With –prof-gen flog added in the compiling command
   ```
   icc –prof-gen –O3 –xAVX your.c –o prog.exe
   ```
2. Run the program
   ```
   ./prog.exe # it will generate a dynamic profile: xx1.dyn
   ```
3. Recompile the code with –prof-use flag
   ```
   icc –prof-use –O3 –xAVX your.c –o prog1.exe
   # it will use the 1234.dyn information
   ```
4. ```
   ./prog.exe # it will generate another dynamic profile: xx2.dyn
   ```
5. Repeat the steps 1 – 4
Code vectorization – vec-report\((n)\)

Vectorizer is enabled for automatically vectorizing the code when -O3, -ax or –x option is specified.

**Arguments** \(n\)

- 0 report no diagnostic information.
- 1 report on vectorized loops (default)
- 2 report on vectorized and non-vectorized loops.
- 3 report on vectorized and non-vectorized loops and any proven or assumed data dependences.
- 4 Tells the vectorizer to report on non-vectorized loops.
- 5 Tells the vectorizer to report on non-vectorized loops and the reason why they were not vectorized.
Multithreading with OpenMP

-openmp

Apply OpenMP directives for shared memory parallelization

Automatic Multithreading

-parallel generate multithreaded code automatically for simple loops with no dependencies.
Tuned Supportive Libraries

Libraries optimized for Intel architecture processors:
  — Intel® Integrated Performance Primitives (Intel® IPP)
  — Intel® Math Kernel Library (Intel® MKL, v10.3 or newer)
    key string: mkl_avx
  — Intel® Threading Building Blocks (Intel® TBB)
  — Intel® MPI Library
    module load impi

Recompiled the needed libraries in the code package with either
-xavx or -axavx
Sandy Bridge Nodes - Multi-core Architecture with large-memory capacity

Use of \texttt{KMP\_AFFINITY} variable

Syntax:

\begin{verbatim}
export KMP\_AFFINITY=${\{modifier}\},${\{type}\}
\end{verbatim}

where \texttt{${\{modifier}\}} is optional

\begin{verbatim}
noverbose   - (default)
verbose    - report the topology map.
\end{verbatim}

\texttt{${\{type}\}} is requested. It accepts value

\begin{verbatim}
none
or compact
or scatter
or disabled
\end{verbatim}
${\text{type}} = \text{none} \ (\text{default})$

Does not bind OpenMP threads to particular thread contexts; however, if the operating system supports affinity, the compiler still uses the OpenMP thread affinity interface to determine machine topology.

Specify $\text{KMP_AFFINITY}=\text{verbose},\text{none}$ to list a machine topology map.
**KMP_AFFINITY=compact**

Assigns the OpenMP thread \(n+1\) to a free thread context as close as possible to where the \(n\) OpenMP thread was placed.
KMP_AFFINITY=scatter

Distributes the threads as evenly as possible across the entire system.
KMP_AFFINITY=disabled

Completely disables the thread affinity interfaces.

permute and offset combinations

For both compact and scatter, permute and offset are allowed.

For more info:
#KMP_AFFINITY_Environment_Variable
Performance profiling

Identifying Hot Spots with VTune Performance Analyzer provide advice at the assembly language level. Where appropriate, pseudo-code provided to suggest the use of highly optimized intrinsics and functions in the Intel® Performance Library Suite.

module load
Performance optimization – first touch

A memory page mapped to local memory of the processor when it is first touched - initialization code

```fortran
integer,parameter :: N=1000000 real*8 A(N), B(N)
A=0.d0
!$OMP parallel do do i = 1, N
B(i) = function ( A(i) ) end do
```

```fortran
integer,parameter :: N=1000000 real*8 A(N), B(N)
!$OMP parallel do schedule(static)
do i = 1, N
A(i)=0.d0
end do
!$OMP parallel do schedule(static)
do i = 1, N
B(i) = function ( A(i) ) end do
```
What performance speed up will be, compared to Itasca?

A factor of 2, SMP applications up to eight cores (Flops: MB/s)
How does the thread affinity affect performance?

Compiled by icc -O3 -xAVX -mcmmodel=large -openmp stream.c

AVX effects not significant

![Graph showing triad performance (MB/s) for different thread configurations and affinity settings. The graph includes data for 'Compact', 'None', and 'Scatter' affinity methods, with performance measured in MB/s for various numbers of nodes and AVX configurations.]
What kind computations perform better with -xavx? Compare AVX with NO, not significant either.

Y = exp(-X)

Performance (MB/s)

4 4 4 4 8 8 8 8 12 12 12 12 16 16 16 16
AVX No No1 axAVX AVX No No1 axAVX AVX No No1 axAVX
What kind computations perform better with -xavx? Compare AVX with NO, Significant!

![Graph showing performance comparison between AVX and NO with different data sizes and AVX modes. The graph plots performance (MB/s) against data size (Y=a * sqrt(X)). The bars represent different compactness and scatter options.](image)
What kind computations perform better with -xavx?

Compare AVX with axAVX, **Significant – AVX-SSE penalty!**

\[ Y = a \times \sqrt{X} \]
Run the job on the right hardware

Memory needs and I/O management
## Configuration of Itasca

<table>
<thead>
<tr>
<th>Queue</th>
<th>Wall clock/-l walltime</th>
<th>Nodes/-l nodes</th>
<th>Cores/-l ppn</th>
<th>memory (mb)/-pmem</th>
<th>Service Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>devel</td>
<td>2:00:00</td>
<td>64</td>
<td>8</td>
<td>2500</td>
<td>debugging and development</td>
</tr>
<tr>
<td>long</td>
<td>48:00:00</td>
<td>17</td>
<td>8</td>
<td>2500</td>
<td>jobs need &gt; 24 hours</td>
</tr>
<tr>
<td>batch</td>
<td>24:00:00</td>
<td>1083</td>
<td>8</td>
<td>2500</td>
<td>parallel jobs SMP and MPI</td>
</tr>
<tr>
<td>oc</td>
<td>24:00:00</td>
<td>4</td>
<td>varied</td>
<td>2500</td>
<td>Serial jobs</td>
</tr>
<tr>
<td>jay</td>
<td>24:00:00</td>
<td>1</td>
<td>varied</td>
<td>7500</td>
<td>Visualization</td>
</tr>
<tr>
<td>sb</td>
<td>24:00:00</td>
<td>35</td>
<td>varied</td>
<td>7500</td>
<td>large memory</td>
</tr>
<tr>
<td>sb128</td>
<td>96:00:00</td>
<td>8</td>
<td>varied</td>
<td>15000</td>
<td>in between</td>
</tr>
<tr>
<td>sb256</td>
<td>96:00:00</td>
<td>8</td>
<td>varied</td>
<td>30000</td>
<td>extremely large memory</td>
</tr>
</tbody>
</table>
I/O management

Use the local /scratch on the compute node for input and output as much as possible.

```bash
pbsdsh -u "cp /home/its/szhang/input /scratch"
mpirun -np 20 ./a.out /scratch/input > /scratch/output
mv /scratch/output /home/it1/username
rm /scratch/input /scratch/output
```

Use /lustre for large scale of I/O operations

```bash
cd /lustre;mkdir your_name
```

Generate multiple directories for parallel I/O operation

```bash
for io in `seq 1 4`
do
  mkdir $io
done
```
Find the optimal MPI implementation
PMPI, IMPI, OMPI
VTune can be a useful tool
Runtime environment setting

Add the following two lines after #PBS commands

```
a1=$(cat $PBS_NODEFILE | sort | uniq)
pdsh -w `echo $a1 | sed 's/ /,/g'` date >& check_node
```

Many env variables can be adjusted
For example:
```
export MPI_MAX_REMSH=16
export MPI_MAX_MPID_WAITING=128
```
User guide manual is in /opt/platform_mpi/doc
Reduce the queuing time

Match the application with the queues
  • Knowing your job well memory, I/O, and time duration
  • Knowing well about the queue configuration
  • Match the application

Useful command
qstat -q
showbf
qsub -l -X -l walltime=2:00:00,mem=2gb,nodes=2
Questions?