How Do I Use the New Sandy Bridge Nodes?

June 21 & 27, 2012
NASA Advanced Supercomputing Division
Outline

• Pleiades Augmentation/Expansion
  – Removal of Some Harpertown and Addition of Sandy Bridge

• Main Differences among the Four Pleiades Processor Types
  – At the Node Level
  – Inside the Sandy Bridge Processor Cores
    * Advanced Vector Extensions (AVX) and Floating Point Operations

• What Do I Need to Do to Run on the Sandy Bridge?
  – Do I Need to Recompile My Code?
  – Performance Comparisons of Some Applications (w. different compiler flags)
  – Do I Need to Change My PBS Script?

• Should I Run on the Sandy Bridge?
  – Availability Consideration
  – Memory Consideration
  – Performance and SBU Rates Consideration
### Pleiades Augmentation/Expansion

<table>
<thead>
<tr>
<th></th>
<th>Harpertown</th>
<th>Nehalem</th>
<th>Westmere</th>
<th>Sandy Bridge</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Racks</td>
<td>91</td>
<td>64</td>
<td>20</td>
<td>74</td>
</tr>
<tr>
<td># of Nodes</td>
<td>5,824</td>
<td>4,096</td>
<td>1,280</td>
<td>4,672</td>
</tr>
<tr>
<td># of Cores</td>
<td>46,592</td>
<td>32,768</td>
<td>10,240</td>
<td>56,064</td>
</tr>
<tr>
<td>Peak TFlops</td>
<td>559</td>
<td>393</td>
<td>120</td>
<td>658</td>
</tr>
</tbody>
</table>

- Total hardware: **182 racks, 11,776 nodes, 126,720 cores**
- Total theoretical peak performance: **1.75 Pflops/s**
### Main Differences among Pleiades Processors

<table>
<thead>
<tr>
<th></th>
<th>Harpertown</th>
<th>Nehalem</th>
<th>Westmere</th>
<th>Sandy Bridge</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Intel Model</strong></td>
<td>E5472</td>
<td>X5570</td>
<td>X5670/X5675*</td>
<td>E5-2670</td>
</tr>
<tr>
<td><strong># of Cores per Node</strong></td>
<td>2 x 4-core = 8</td>
<td>2 x 4-core = 8</td>
<td>2 x 6-core = 12</td>
<td>2 x 8-core = 16</td>
</tr>
<tr>
<td><strong>CPU-Clock</strong></td>
<td>3.0 GHz</td>
<td>2.93 GHz</td>
<td>2.93/3.06 GHz</td>
<td>2.6 GHz</td>
</tr>
<tr>
<td><em><em>DP FPs</em> per cycle per core</em>*</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td><strong>Largest Cache</strong></td>
<td>6 MB*/2 cores</td>
<td>8 MB/4 cores</td>
<td>12 MB/6 cores</td>
<td>20 MB/8 cores</td>
</tr>
<tr>
<td><strong>Memory per node</strong></td>
<td>8 GB*</td>
<td>24 GB</td>
<td>24 GB*</td>
<td>32 GB*</td>
</tr>
<tr>
<td><strong>Memory per Core</strong></td>
<td>1 GB</td>
<td>3 GB</td>
<td>2 GB</td>
<td>2 GB</td>
</tr>
<tr>
<td><strong>Memory Bandwidth per socket</strong></td>
<td>25.6 GB/s read, 12.8 GB/s write</td>
<td>32 GB/s</td>
<td>32 GB/s</td>
<td>51.2 GB/s</td>
</tr>
<tr>
<td><strong>QPI between sockets</strong></td>
<td>Not applied</td>
<td>25.6 GB/s</td>
<td>25.6 GB/s</td>
<td>32 GB/s</td>
</tr>
<tr>
<td><strong>IB between nodes &amp; to Lustre</strong></td>
<td>4x DDR (4 x 5 Gb/s)</td>
<td>4x DDR/QDR* (4 x 10 Gb/s)</td>
<td>4x QDR (4 x 10 Gb/s)</td>
<td>4x FDR (4 x 14 Gb/s)</td>
</tr>
</tbody>
</table>

- Westmere racks 221 & 222 use X5675; rack 219 also includes Nvidia GPUs
- DP FPs: Double Precision Floating Point Operations per second
- Largest cache in Harpertown is L2; the others are L3
- Bigmem nodes: Harpertown nodes in rack 32 have 16 GB/node; 17 Westmere nodes have 48 GB/node; 4 Westmere nodes have 96 GB/node; Some Sandy Bridge nodes may have more memory in the near future
- The Nehalem racks use DDR Host Channel Adapters (HCAs) and QDR switches

[Question? Use the Webex chat facility to ask the Host]
Sandy Bridge Node Configuration

Physical id= 0

Processor id 0/16
- core
- L2

Processor id 1/17
- core
- L2

Processor id 2/18
- core
- L2

Processor id 3/19
- core
- L2

20 MB L3 Cache

L2 core

L2 core

L2 core

L2 core

Memory Controller

I/O Controller

QuickPath Interconnect

1600 MHz

16 GB DDR3 Memory

51.2 GB/s read/write

Physical id= 1

Processor id 8/24
- core
- L2

Processor id 9/25
- core
- L2

Processor id 10/26
- core
- L2

Processor id 11/27
- core
- L2

20 MB L3 Cache

L2 core

L2 core

L2 core

L2 core

QuickPath Interconnect

I/O Controller

Memory Controller

32 GB/s @8.0 GT/s

16 GB DDR3 Memory

51.2 GB/s read/write

Question? Use the Webex chat facility to ask the Host
Improvements of Sandy Bridge Node over Westmere Node (not considering the cores)

- Larger L3 cache (2.5MB/core vs 2.0MB/core)
- Higher memory speed and bandwidth (1600 MHz vs 1333 MHz; 4 channels vs 3 channels)
- Higher Quick Path Interconnect speed/bandwidth
- Faster communication between nodes via FDR vs QDR

An application can perform better due to these improvements alone.

Next, what has changed inside the core?
Floating Point Operations inside a Core

Har/Neh/Wes use 128 bits XMM FP register (SSE)

Intel AVX widens the register length to 256 bits

Sandy Bridge uses 256 bits YMM FP register (AVX)

Sample instruction in a loop: \( X(i) = X(i) + Y(i) \)

<table>
<thead>
<tr>
<th>Scalar Mode</th>
<th>Vectorization with SSE (Streaming SIMD Extensions)</th>
<th>Vectorization with AVX (Advanced Vector Extensions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X + Y</td>
<td>X1 ( + ) X0</td>
<td>X3 ( + ) X2 ( + ) X1 ( + ) X0</td>
</tr>
<tr>
<td></td>
<td>Y1 ( = ) Y0</td>
<td>Y3 ( = ) Y2 ( + ) Y1 ( + ) Y0</td>
</tr>
<tr>
<td></td>
<td>X1  ( + ) Y1 ( = ) X0  ( + ) Y0</td>
<td>X3+Y3 ( = ) X2+Y2 ( = ) X1+Y1 ( + ) X0+Y0</td>
</tr>
</tbody>
</table>

Vectorization of a loop = unrolling the loop so that it can take advantage of packed SIMD instructions to perform the same operation on multiple data in a single instruction.
Floating Point Operations inside a Core (cont’d)

Number of SSE instructions: SSE - 70, SSE2 - 144, SSE3 - 13, SSSE3 - 16, SSE4.1 - 47, SSE4.2 – 7 (compiler flags: -xSSE4.1 or –xSSE4.2)
SSE includes instructions for arithmetic, logic, compare, convert, load/store, etc. operations, not all are related to floating point operations or use the XMM registers.

Number of AVX instructions: 12 (compiler flag: –xAVX)

- In each core, there are 16 floating point registers and 2 floating point functional units

- If a code is well vectorized with data pipelined in the 16 registers, and both functional units are busy, it can achieve
  - maximum 4 double precision Flops/cycle/core or 8 single precision Flops on Har/Neh/Wes with SSE
  - maximum 8 double precision Flops/cycle/core or 16 single precision Flops on Sandy Bridge with AVX

Problem: many user codes do not achieve these maximum Flop rates

- If your code gets more Flops per cycle on Sandy Bridge than on the other three processor types, even with lower clock speeds (2.6 GHz vs 3.0/2.93 GHz), the time spent on floating point operations may be shorter
Taking Advantage of AVX

- Use compiler flag `-vec-report2` (need `-O2` and above) to get reports on why loops are not vectorized
- Modify source code to allow more vectorization
  (only inner loop can be vectorized; unless inlined, avoid function/subroutine calls in loops; no branches, number of loop iterations must be known; avoid non-unit stride or indirect addressing; no dependency; make code 32 bytes aligned, etc.)
- Use Intel MKL library (`-mkl`) which continues to be optimized for newer generations of Intel architecture (including AVX)
- References about vectorization and AVX
  - Requirements for Vectorizable Loops http://software.intel.com/en-us/articles/requirements-for-vectorizable-loops/
  - Pdf file “Compiling for the Intel 2nd Generation Core processor family and the Intel AVX instruction set” http://software.intel.com/file/34217/
Do I Need to Recompile My Code?

• No. If you do not care about performance, your existing executable that ran on Har/Neh/Wes should work on Sandy Bridge.

• Yes. If you want to explore getting better performances. We recommend:
  - Use latest Intel version 12 compiler on Pleiades
  - Experiment with –O2 vs –O3 and –ip (or –ipo if you have multiple source files)
    Note: Vectorization is enabled at –O2 and above; -O3 allows more loop optimization than –O2
  - Experiment with adding –xAVX (code runs on SAN only)
  - If you choose to use –xAVX, apply it to all source files
    (there is performance penalty with functions compiled with –xAVX, -xSSEEn calling each other)
  - Experiment with –axAVX –xSSE4.1 (code runs on Har/Neh/Wes/San; gives both SSE and AVX code paths with SSE4.1 as the default; Which code path is used is determined at runtime)

• Why choosing Intel version 12 over version 11?
  - Version 11.1 accepts –xAVX (though not documented in man page) but performance may not be as good as version 12. Version 11.0 does not accept -xAVX
  - MKL 10.3 used in version 12 compilers includes more AVX optimization (dgemm/sgemm, all BLAS level 3 functions, …) than MKL 10.2 used in version 11 compilers
  - Even without –xAVX, version 12 may provide more optimization for your code

• Whatever combination you choose, verify correctness is important!!
## Effects of –x or –ax Choices at Run Time

<table>
<thead>
<tr>
<th>Choice of –x or -ax</th>
<th>Harpertown</th>
<th>Nehalem</th>
<th>Westmere</th>
<th>Sandy Bridge</th>
</tr>
</thead>
<tbody>
<tr>
<td>None (default to -mSSE2)</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>-xSSE4.1</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>-xSSE4.2</td>
<td>Abort</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>-xAVX</td>
<td>Abort</td>
<td>Abort</td>
<td>Abort</td>
<td>OK</td>
</tr>
<tr>
<td>-axAVX –xSSE4.1</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
</tbody>
</table>

**Fatal Error:** This program was not built to run on the processor in your system. The allowed processors are: Intel(R) processors with SSE4.2 and POPCNT instructions support. [Note: POPCNT: Population count (count number of bits set to 1)]

**Fatal Error:** This program was not built to run in your system. Please verify that both the operating system and the processor support Intel(R) AVX.
Performance (in sec) of a Simple Matrix Multiply

program main
parameter (nmax=1000)
real (kind=8), dimension(nmax,nmax):: a, b, c
integer (kind=8) :: t0, t1, rate
a = 1.0
b = 2.0
c = 0.0
call system_clock(t0, rate)
do j = 1, nmax
  do k = 1, nmax
    do i = 1, nmax
      c(i,j) = c(i,j) + a(i,k) * b(k,j)
    end do
  end do
end do
call system_clock(t1, rate)
print *, 'c(100, 201) = ', c(100,201)
print *, 'time = ', float(t1-t0)/rate
stop
end

Version 12.1.0 compiler used: comp-intel/2011.7.256
Performance of a Simple Matrix Multiply (cont’d)

- Time: O3 < O2 < O1 on all processor types
- –xSSE4.1 or –xSSE4.2 does not improve performance on all processor types (We saw this behavior for many codes)
- –xAVX does improve performance on Sandy Bridge
- –xAVX –xSSE4.1 gives good performance (as good as –xSSE4.2 or –xSSE4.1 on Har/Neh/Wes and as good as –xAVX on San) and also allows the executable to run on all processor types
- Using dgemm in MKL performs better than original code
- Performance of dgemm is controlled by how MKL was built by Intel. It is not sensitive to your choice of compiler flags

**Warning:** This example demonstrates an ideal case. Performance of your code may deviate from this observation. You should experiment yourself!!
# Preliminary Performances (in sec) of SBU & some NPB Benchmarks

<table>
<thead>
<tr>
<th>V12 with –O3</th>
<th>Cores Used</th>
<th>Wes</th>
<th>San (no –xAVX)</th>
<th>San (with –xAVX)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENZO</td>
<td>240</td>
<td>1925</td>
<td>1854</td>
<td>1637</td>
</tr>
<tr>
<td>FUN3D (–O2)</td>
<td>960</td>
<td>1734</td>
<td>1438</td>
<td>1449</td>
</tr>
<tr>
<td>GEOS-5</td>
<td>1176</td>
<td>2096</td>
<td>1327</td>
<td>1235</td>
</tr>
<tr>
<td>OVERFLOW</td>
<td>480</td>
<td>1786</td>
<td>1200</td>
<td>1154</td>
</tr>
<tr>
<td>USM3D</td>
<td>480</td>
<td>1802</td>
<td>1583</td>
<td>1545</td>
</tr>
<tr>
<td>WRF</td>
<td>384</td>
<td>2036</td>
<td>1540</td>
<td>1499</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V12 with –O3</th>
<th>Cores Used</th>
<th>Wes</th>
<th>San (no -xAVX)</th>
<th>San (with –xAVX)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT.C</td>
<td>16</td>
<td>102.2</td>
<td>92.7</td>
<td>84.6</td>
</tr>
<tr>
<td>CG.C</td>
<td>16</td>
<td>26.9</td>
<td>15.8</td>
<td>16.8</td>
</tr>
<tr>
<td>FT.C</td>
<td>16</td>
<td>33.9</td>
<td>19.3</td>
<td>18.4</td>
</tr>
</tbody>
</table>

Source of data: ENZO: S. Heistand, FUN3D, OVERFLOW, USM3D: J. Djomehri, GEOS-5, WRF: S. Cheung, NPB: H. Jin. Turbo Boost was ON.

Question? Use the Webex chat facility to ask the Host
Do I Need to Change My PBS Script?

- Change the number of nodes, ncpus and model

  ```
  #PBS -lselect=4:ncpus=12:model=wes
  #PBS -lselect=3:ncpus=16:model=san
  ```

- Change the MPT version for MPI applications
  
  - Support for Mellanox InfiniBand FDR devices starts in MPT 2.05. MPT 2.06 has additional bug fixes and is recommended for cross-node jobs

    ```
    module load comp-intel/2011.7.256 mpi-sgi/mpt.2.06a67 (may change)
    ```
  
  - If you load mpi.2.04.10789 or earlier versions, job will not run:

    Get this message: ```Sandy Bridge nodes need mpt 2.06 or later```

- For better performances, may use mbind.x to pin processes/threads if not all cores are used

  ```
  #PBS -lselect=3:ncpus=12:model=san
  
  mpiexec -np 36 mbind.x -cs -n12 -v $HOME/a.out > /nobackup/user
  ```

  # info about mbind.x: [http://www.nas.nasa.gov/hecc/support/kb/entry/288](http://www.nas.nasa.gov/hecc/support/kb/entry/288)
Should My Job Run on Sandy Bridge?

**Availability Consideration**
- 2 racks (2,304 cores) set aside for devel queue; 22 racks for normal, long, debug queues
  
  \[
  \text{qsub} \quad -q \text{ devel@pbspl3 your_job_script} \quad ; \quad \text{qsub} \quad -q \text{ long[@pbspl1] your_job_script}
  \]
- Use `qstat –au foo [@pbspl3]` to check if there are free SAN nodes
- Use `qstat –i –W o=+model,mission [@pbspl3]` to check resources requested for queued jobs

**Memory Consideration**
- 32 GB/node; some apps that got OOMs on Har/Neh/Wes may run on San
- Sandy Bridge bigmem nodes (128/256 GB) may be available in the near future

**Performance and SBU Rates Consideration**
- Most applications should run faster (even without the use of –xAVX) on Sandy Bridge, but you should check this for your own application
- Is it cheaper to run on Sandy Bridge than others?

Check if \# of San nodes used \times san \text{ wall_time} \times 1.65 < \# of Wes nodes used \times wes \text{ wall_time} \times 1

<table>
<thead>
<tr>
<th></th>
<th>San</th>
<th>Wes</th>
<th>Neh</th>
<th>Har</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores/node</td>
<td>16</td>
<td>12</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>SBU Rate</td>
<td>1.65* (may change)</td>
<td>1</td>
<td>0.8</td>
<td>0.45</td>
</tr>
</tbody>
</table>

*Turbo Boost OFF
Summary

- The Sandy Bridge processor is significantly different from Har/Neh/Wes
- Recompilation of your source code with v. 12 compiler is recommended.
  (use –axAVX –xSSE4.1 if you want an executable that works on all)
- Use MKL libraries (-mkl) when possible
- Your code may or may not benefit from the AVX technology
- Most codes should benefit from other improvements in Sandy Bridge
  (larger L3, higher memory bandwidth, faster interconnect)
- Checking correctness is important
- /nobackup file systems are accessible from Sandy Bridge nodes
- Minimal modification of your PBS script is required for Sandy Bridge
- Devel (pbspl3), normal, debug, long (pbspl1) queues are available now
- Check performance and SBU usage to see if you should run on Sandy Bridge
  (acct_query –pall –call –ujsmith –olow for jobs finished today)

There may be IB or Lustre issues, but SGI/NAS continue to work on stabilizing them. Contact NAS Help Desk if you need further assistance.

http://www.nas.nasa.gov/hecc/support/kb/Preparing-to-Run-on-Pleiades-Sandy-Bridge-Nodes_322.html