Technical Briefing At CESCA
Gaussian

Roberto Gomperts
Senior Principal Scientist
SGI
Thanks To...

- John Baron
- Huiyu Feng
- Chao Ma
- Daniel Thomas
- Jia Liu
- Michael Raymond
- Karl Feind
- Matthias Fouquet-Lapar
- Martin Hilgeman
Topics

- Parallelism Models in Gaussian
- %NProc and %Mem
- Recognizing problems in the output
- Boundaries of parallelism
- Miscellaneous Considerations
- Placement
- “Cache” Sizes
- Dynamic Load Distribution
Parallelism Models in Gaussian

- Distributed Memory (Clusters)
  - Based on Linda (SCA - Gaussian)
  - \( %\text{NProcLinda}=n \)
  - a subset of what is available in shared memory

- Shared Memory (Any Multiprocessor Platform)
  - Based on OpenMP
  - \( %\text{NProcShared}=n \) or \( %\text{Nproc}=-n \)

- Possible to combine: Linda between nodes, OpenMP within a node
Gaussian: %Nproc vs. %Mem

“Traditional” Shared memory

“Traditional” Distributed memory - similar in Linda

Gaussian Shared memory with OpenMP
Watch Out for

- In direct methods:
  - “PrsmSu: requested number of processors reduced to: 3 ShMem 1 Linda.
  - “Number of processors reduced to ...”

- In MP2
  - “JobTyp=1 Pass 1: I= 1 to 9 NPSUSe= 8 ...”

- For SCF/DFT Methods:
  - Mem = Mem1 + Mem1 * 0.75 * (Nproc-1)

- For PostSCF Methods + Freq:
  - Mem = Mem1 * Nproc
Eficiencia y la Ley de Amdahl

Los beneficios se reducen:
Conflicto entre el deseo de emplear más CPUs y el “coste” asociado

\[ e = \frac{a}{n} \]
\[ e = \frac{1}{p + n(1-p)} \]

- \( e \): eficiencia
- \( n \): número de procesadores
- \( p \): fracción paralela
## Our Experience

<table>
<thead>
<tr>
<th>Method</th>
<th>Amount of Parallelism (50% Efficiency)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HF/DFT</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>48p (97.8%)</td>
</tr>
<tr>
<td></td>
<td>Seen 120p (99.1%)</td>
</tr>
<tr>
<td></td>
<td>36-60p (97.1-98.3%)</td>
</tr>
<tr>
<td></td>
<td>48p (97.8%)</td>
</tr>
<tr>
<td><strong>MP2</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>36-48p (97.1-97.8%)</td>
</tr>
<tr>
<td></td>
<td>36-48p (97.1-97.8%)</td>
</tr>
<tr>
<td></td>
<td>24p (95.6%)</td>
</tr>
<tr>
<td><strong>CCSD(t)</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations: nihil</td>
</tr>
<tr>
<td></td>
<td>Triples: 24-36p (95-97%)</td>
</tr>
</tbody>
</table>
Miscellaneous

- **Freq freqmem**
  - When running frequency calculations use `freqmem` for optimal amount of memory on 1 processor. Then use `Mem1*nProc` memory

- **MP2: conventional, SemiDirect, FullDirect, InCore**

- **Integral Transformations MP4, CCSD(t), QCISD(t)**
  - Difficult to determine optimal memory usage
  - **My experience:**
    - Often SemiDirect with few passes (>1) is best. This can be controlled via iOP or better with MaxDisk
    - If enough memory present FullDirect can be as efficient, again # of passes > 1
    - In general I have found InCore to be least efficient
    - Never use conventional
Placement: “first touch”

- When running on SGI Altix processor affinity and memory affinity can be important
  - Keep caches ‘hot’
  - Diminish traffic in communication fabric
  - “first touch” placement policy
Placement: dplace and MP_BIND

- SGI’s NUMAtools
  - dplace (man dplace): establishes a “container” and “pins” processes
  - Just pinning the processes
    dplace -s1 g09 < input >& output &
  - Explicit pinning of processes
    dplace -s1 -c $CPULIST g09 < input >& output &
    Environmental variable CPULIST of the form: 4-7 or 0-8:2
  - Watch out for CPUsets

- PGI’s Environment
  - setenv MP_BIND “yes”
    setenv MP_BLIST $CPULIST
    Environmental variable CPULIST of the form: 4,5,6,..
    No intervals or jump notation
Default.Route

- Default.Route file handy to set up
  - System wide defaults (located in $g09root/g09)
  - Specific values (located in working directory)
  - Typical: -#-Maxdisk=xxGB

- CacheSize en IA64
  - Use System-wide Default.Route with:
    -#-CacheSize=size
    size= 1+L3/16  L3: size of L3 cache in bytes

- CacheSize en x86_64
  - Best value for most cases: 1 MB (8Mw)
Multi-core Diagram
Effect of Cache Size

- Check using: `testrt hf`

- Many experiments showed that on x86_64 processors (NHM and WSM) that the optimal value is close to the value used in the distribution binaries (1MB)

- Experiments included:
  - Increasing the size of runs on 1 core (make use of the “expanded” size of the shared caches on a socket)
  - Diminishing the size when using more cores per socket
  - Using the same size cache but using just a few cores on the socket
Gaussian - Effects of CacheSize

The NHM (X5570) microprocessor shares an 8 MB last level cache over 4 cores on 1 socket. So when running on 8 cores or when using an “ss” placement with 4 cores it is expected that “2MB” is the optimal CacheSize. This is also what I found (“ss” results are not shown here)

When using 1 core or 2 cores with an “nn” placement we found the “4MB” to be optimal, instead of the expect “8MB”. The differences are barely significant

On 4 cores with “nn” placement the optimal CacheSize setting is, as expected, “4MB”

For the rest of the experiments with Gaussian we used “4MB” for runs on 1 to 4 cores (with “nn” placement) and “2MB” for runs on 8 cores.

We compared “1MB” with “2MB” in runs on 16 cores with SMT no significant performance differences were seen between these two settings
Issues Around Dynamic Load Balancing

- Gaussian implements in many paths a Dynamic Load Distribution scheme
  - Based on keeping two sets of counters: a Global (Shared) counter and a Local (Private) counter
  - The presence of a Global counter means that Critical Regions are required
  - Since the counters are placed in one of the most inner loops means that it can have an adverse effect on performance

- Solution/Workaround
  - Use chunks to diminish the number of Critical Regions
    - Implemented in G09 rev B.01
  - Use keyword: Int=NoDynPar
Testing Int=NoDynPar

- Test: $\alpha$-pinenefreqb
- Number of cores: 48
- B.01 shows the improvements that can be attributed to Dynamic Load Distribution
- A.02 shows the degradation caused by excessive locking

<table>
<thead>
<tr>
<th>Revision</th>
<th>NoDynPar?</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.02</td>
<td>Default</td>
<td>679.46</td>
</tr>
<tr>
<td>A.02</td>
<td>NoDynPar</td>
<td>520.57</td>
</tr>
<tr>
<td>B.01</td>
<td>Default</td>
<td>369.52</td>
</tr>
<tr>
<td>B.01</td>
<td>NoDynPar</td>
<td>499.57</td>
</tr>
</tbody>
</table>
Comparing Gaussian’s 09 Performance of revision B.01 with revision A.02

Roberto Gomperts
Topics

- Computer Systems
- Gaussian Software
- Case Study effect of Remote Latencies (OpenMP; Gaussian 03)
- Summary
- Gaussian
  - Tests
  - Description of Experiments
  - Performance of
    - SGI Altix 4700
    - SGI Altix UV1000
Computer Systems

- **SGI Altix UV1000 NHM-EX 7542**
  - 32 2-sockets HUB 2.0; each socket has an 6-core Intel Xeon X7542 (NHM-EX) 2.66 GHz, 18 MB cache (total 384 cores)
  - Total Mem: 2 TB  Speed: 1067 MHz (0.9 ns)
  - SLES11SP0 OS, SGI ProPack 7SP2
  - HyperThreading not available
  - TurboBoost not enabled

- **SGI Altix 4700 “Montvale”**.
  - Itanium dual-core processors: 9150M Itanium @ 1.66GHz 24MB cache (12MB/core); 667 MHz FSB (total 256 cores)
  - Total Mem: 512 GB
  - SLES10SP2 OS, SGI ProPack 6SP3
Gaussian 09

- **Gaussian 09 rev. A.02**
  - Binaries supplied by Gaussian, Inc.
  - Itanium
    - Intel Compilers 11.0.083
    - MKL 10.1.1.019
  - X86_64
    - Portland Group Compilers (F77) 8.0-6
      - These compilers support up to a maximum of 64 threads in OpenMP
    - Libraries: Atlas (pre-compiled by Gaussian, Inc.)

- **Gaussian 09 rev. B.01**
  - Binaries supplied by Gaussian, Inc.
  - Itanium
    - Intel Compilers 11.1.072
    - MKL 10.2.5.035
  - X86_64
    - Portland Group Compilers (F77) 10.5
    - Libraries: Atlas (pre-compiled by Gaussian, Inc.)

- When running on more than one node of a cluster with Linda it is not possible to place/pin the threads on remote hosts. We developed at SGI a mechanism (*sgibind*) that allows us achieve this performance enhancing feature. The data reported in this report makes use of *sgibind*
Scalability of Gaussian 03 on SGI Altix: The Importance of Data Locality on CC-NUMA Architecture

Roberto Gomperts¹, Michael Frisch², Jean-Pierre Panziera¹

¹ SGI
² Gaussian, Inc
top - 14:32:46 up 5 days,  1:05, 10 users,  load average: 22.86, 21.96, 13.56
Tasks: 924 total,  33 running, 891 sleeping,  0 stopped,  0 zombie
Cpu(s):  2.0%us,  0.0%sy,  0.0%ni,  97.9%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  144829024k total,  5608320k used, 139220704k free,  12480k buffers
Swap: 195417776k total,  0k used, 195417776k free,  774048k cached

PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM   TIME  P COMMAND
216786 chem      15   0  5200 2768 1632 R   14  0.0   0:00 42 top
216693 chem      25   0 35.6g 2.7g 8016 R  100  1.9  10:39 35 11002.exe
216692 chem      25   0 35.6g 2.7g 8016 R  100  1.9  10:21 34 11002.exe
216691 chem      25   0 35.6g 2.7g 8016 R  100  1.9  10:41 33 11002.exe
216690 chem      25   0 35.6g 2.7g 8016 R  100  1.9  10:14 32 11002.exe
216689 chem      25   0 35.6g 2.7g 8016 R  99  1.9  10:36 31 11002.exe
216688 chem      25   0 35.6g 2.7g 8016 R  99  1.9  10:22 30 11002.exe
216687 chem      25   0 35.6g 2.7g 8016 R  99  1.9  10:43 29 11002.exe
...
Gaussian’s Parallelism and Memory Usage Model

- Two Parallelization Models
  - Shared Memory (OpenMP)
  - Distributed Memory (Linda)
- Possible to Combine Hybrid/Hierarchical
- Linda Parallelism is a subset of OpenMP

Gaussian Shared memory with OpenMP

First-touch Memory Placement Policy
OpenMP Parallelization of Gaussian: Basic Algorithm

Subroutine CPHFdriver
allocate scratch(LARGE)
FA(:,:,:,:) = 0
do while(iterative_solution)
  Call DirSCF(LARGE,scratch,FA)
  Call Update(FA)
enddo

Subroutine DirSCF(LARGE,scratch,FA)
do while(.not. All_ijkl)
  Call CalcInt(LARGE,scratch)
  Call InFock(LARGE,scratch,FA)
enddo

Subroutine CalcInt(LARGE,scratch)
do while(FitInLarge)
  do i,j,k,l
    scratch(i,j,k,l) = f(i,j,k,l)
  enddo
enddo

Subroutine InFock(LARGE,scratch,FA)
do i= 1,LARGE
  do j = 1,nFock
    FA(j,ij,kl) = FA(j,ij,kl) +
    c(j,ij,kl)*scratch(i,j,k,l)
  enddo
enddo
Subroutine CPHFdriver
allocate scratch(LARGE)
FA(:,:, :) = 0
do while( iterative_solution)
   Call DirSCF(LARGE, scratch, FA)
   Call Update(FA)
enddo

Subroutine CPHFdriver
allocate scratch(LARGE)
allocate FA_p(:,:, :, MAXPROC-1)
FA(:,:, :) = 0
sl = LARGE/np; lst_sl = LARGE – sl*(np-1)
do while( iterative_solution)
   C$OMP Parallel DO Private(ip)
   do ip=1,np
      if(ip==np) then
         Call DirSCF(lst_sl, scratch(1+sl*(ip-1)), FA, ip, np)
      else
         FA(:,:, :, ip) = 0
         Call DirSCF(LARGE, scratch(1, sl*(ip-1)),
                    FA_p(1, 1, 1, ip), ip, np)
      endif
   enddo
   do ip=1,np-1
      Call Daxpy(ij*kl, 1.0d0, FA_p(1, 1, 1, ip), 1, FA, 1)
   enddo
   Call Update(FA)
enddo
Original Code: Average Latencies and Cycle Counts

- Four threads per node
  - Load Unbalance
  - Large latencies (cycles)
Hot Spot: Routine Dgst01

Do 500 IShC = 1, NShCom
   IJ = LookLT(C4IndR(IShC,1)+IJOff)
   KL = LookLT(C4IndR(IShC,6)+KLOff)
   R1IJKL = C4RS(IShC,IRS1,1)*( C4ERI(IShC,IJKL)
     $           - FactX*(C4ERI(IShC,IKJL)+C4ERI(IShC,ILJK)) )
   C
   If(Abs(R1IJKL).ge.CutOff) then
      Do 10 IMat = 1, NMatS
      10       FA(IMat,IJ) = FA(IMat,IJ) + DA(IMat,KL)*R1IJKL
      Do 20 IMat = 1, NMatS
      20       FA(IMat,KL) = FA(IMat,KL) + DA(IMat,IJ)*R1IJKL
      endIf
   [Similar If/endIf constructs as above]
500 Continue

Overall Nesting level 5
Parallelization of Gaussian: Modifications for Local DA

Subroutine CPHFdriver
allocate scratch(LARGE)
allocation FA_p(:,:,MAXPROC-1)
FA(:,:,0) = 0
sl = LARGE/np; lst_sl = LARGE - sl*(np-1)
do while(iterative_solution)
C$OMP Parallel DO Private(ip)
do ip = 1,np
   if(ip == np) then
      Call DirSCF(lst_sl, scratch(1 + sl*(ip-1)), FA, ip, np)
   else
      FA(:,:,ip) = 0
      Call DirSCF(LARGE, scratch(1+sl*(ip-1)),
                  FA_p(1,1,1,ip), ip, np)
   endif
endo
endo

Call Update(FA)
endo

Subroutine CPHFdriver
allocate scratch(LARGE)
allocation FA_p(:,:,MAXPROC-1)
FA(:,:,0) = 0
sl = LARGE/np; lst_sl = LARGE - sl*(np-1)
do while(iterative_solution)
C$OMP Parallel DO Private(ip, DA_p)
do ip = 1,np
   if(ip == np) then
      Call DirSCF(lst_sl, scratch(1 + sl*(ip-1)), FA, ip, np)
   else
      FA(:,:,ip) = 0
      Call DirSCF(LARGE, scratch(1+sl*(ip-1)),
                  FA_p(1,1,1,ip), ip, np, DA_p)
   endif
endo
endo

Call Update(FA)
endo
Local DA Code: Latencies

- **Dramatic reduction in Average FSB relative Memory Latency**
  - Local: from ~470 ns to ~320 ns
  - Remote: from ~950 ns to ~370 ns

- **Elapsed Time**
  - Original code: 1140 s
  - Local DA code: 670 s
Memory to Cores Mapping

Original
1140 sec

Intermediate (round-robin)
800 sec

Local DA
670 sec
Conclusions

- In a CC-NUMA architecture performance bottlenecks can be found when many nodes have very frequent and almost simultaneous (read-only) access to the same memory structure on one node.
- Spreading the memory being accessed somewhat alleviates the problem.
- Best solution is to localize (by replication) the read-only data structures.
- SGI’s Histx Performance Analysis tools suite offer an efficient way to diagnose the problem and pin-point the hot-spots.
## Gaussian – Tests

<table>
<thead>
<tr>
<th>Test</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apinehfb</td>
<td>C&lt;sub&gt;10&lt;/sub&gt;H&lt;sub&gt;16&lt;/sub&gt;, hf/6-311++G(3df,3pd) scf=tight, 678 basis functions</td>
</tr>
<tr>
<td>apinedftb</td>
<td>C&lt;sub&gt;10&lt;/sub&gt;H&lt;sub&gt;16&lt;/sub&gt;, b3li/6-311++G(3df,3pd) scf=tight, 678 basis functions</td>
</tr>
<tr>
<td>Taxolb</td>
<td>B3Lyp single point calculation of taxol: C&lt;sub&gt;47&lt;/sub&gt;H&lt;sub&gt;51&lt;/sub&gt;NO&lt;sub&gt;14&lt;/sub&gt;; Geometry from test385; C1 symmetry, cc-pVDZ basis set, 1123 basis functions</td>
</tr>
<tr>
<td>Ala17d</td>
<td>B3Lyp single point calculation of Ala17 C&lt;sub&gt;53&lt;/sub&gt;H&lt;sub&gt;90&lt;/sub&gt;N&lt;sub&gt;18&lt;/sub&gt;O&lt;sub&gt;18&lt;/sub&gt;; Geometry from Ginc; C1 symmetry, 6-31G(d) basis set, 1515 basis functions, Int=Ultrafine, scf=novaracc</td>
</tr>
<tr>
<td>Test397</td>
<td>Valinomycin.Test from Gaussian’s Inc. QA-suite. RB3LYP/3-21G Force calculation of C&lt;sub&gt;54&lt;/sub&gt;H&lt;sub&gt;90&lt;/sub&gt;N&lt;sub&gt;6&lt;/sub&gt;O&lt;sub&gt;18&lt;/sub&gt;, C1 symmetry, 882 basis functions</td>
</tr>
<tr>
<td>rkest397</td>
<td>Valinomycin.Geometry from test397 from Gaussian’s Inc. QA-suite. RB3LYP/6-31g** Force calculation of C&lt;sub&gt;54&lt;/sub&gt;H&lt;sub&gt;90&lt;/sub&gt;N&lt;sub&gt;6&lt;/sub&gt;O&lt;sub&gt;18&lt;/sub&gt;, C1 symmetry, 1620 basis functions</td>
</tr>
<tr>
<td>apinenefreqb</td>
<td>C&lt;sub&gt;10&lt;/sub&gt;H&lt;sub&gt;16&lt;/sub&gt;, RB3LYP/6-311g(df,p), 346 basis functions, Frequency calculation</td>
</tr>
<tr>
<td>TATBtdst</td>
<td>TATB TD-DFT calculation with Single and Triple excitations. Geometry from test178 from Gaussian’s Inc. QA-suite (TATB: C&lt;sub&gt;6&lt;/sub&gt;H&lt;sub&gt;5&lt;/sub&gt;N&lt;sub&gt;6&lt;/sub&gt;O&lt;sub&gt;6&lt;/sub&gt;). b3li TD=(nStates=8,50-50)/6-311++G(d,p) SCF=noIncore, C2v symmetry, 438 basis functions</td>
</tr>
<tr>
<td>TATBcis</td>
<td>TATB CIS calculation. Geometry from test178 from Gaussian’s Inc. QA-suite. CIS=Direct, SCF=NoIncore calculation of C&lt;sub&gt;6&lt;/sub&gt;H&lt;sub&gt;5&lt;/sub&gt;N&lt;sub&gt;6&lt;/sub&gt;O&lt;sub&gt;6&lt;/sub&gt;, C2v symmetry, 6-311g(2d,p) basis set, 450 basis functions</td>
</tr>
<tr>
<td>hismp2fd</td>
<td>Full Direct MP2 calculation of Histidine-H+ complex: C&lt;sub&gt;6&lt;/sub&gt;H&lt;sub&gt;9&lt;/sub&gt;N&lt;sub&gt;3&lt;/sub&gt;O&lt;sub&gt;2&lt;/sub&gt;, Optimized geometry (output) of test310; C1 symmetry, MP2=FullDirect/6-311++G(3df,3pd) SCF=noIncore, 591 basis functions</td>
</tr>
<tr>
<td>hismp4</td>
<td>MP4 calculation of Histidine-H+ complex: C&lt;sub&gt;6&lt;/sub&gt;H&lt;sub&gt;9&lt;/sub&gt;N&lt;sub&gt;3&lt;/sub&gt;O&lt;sub&gt;2&lt;/sub&gt;, Optimized geometry (output) of test310; C1 symmetry, MP4/6-31G(d) SCF=noIncore, 183 basis functions</td>
</tr>
</tbody>
</table>
Performance Reporting

- For each machine we present first the relative performance of the two revisions based on per-core and on “per-node” timings
  - A “node” (blade) is defined as
    - A “node” on an SGI Altix 4700 has 2 sockets with 2 cores each (total of 4 cores per “node”)
    - The SGI Altix UV1000 “node” has 2 sockets with 6 cores each (total of 12 cores per “node”)

- The parallel performance graphs show the performance (Jobs/day) of our timing suite
- The graphs go up to (around) the number of cores where we still see 50% parallel efficiency (on a per core basis), or the maximum size of the smallest computer system, or the limit is reached of the compiler
Performance Reporting

- The test cases have been grouped as follows:
  - Small molecules and large basis sets
  - Large molecules with small basis sets
  - Force calculations
  - Frequency and TD-DFT calculations
  - CIS calculation
  - Post-SCF: MP2 and MP4

- The principal improvement for parallelism in rev. B.01 consists in addressing the “excessive shared-memory locks” issue
  - This problem was first identified when running on SGI Altix 4700 Frequency and TD-DFT calculations on more than 24 cores
  - The “Fock-matrix” related Direct Calculations paths in the code are most affected by the improvements
  - No effect is expected for Post-SCF calculations
On a single core there is essentially no performance difference between the two revisions.

Since the SGI Altix 4700 "nodes" (4 cores) are relatively small, there is no noticeable effect of the improvements in parallelism between the two revisions. On average rev. B.01 is 4% faster than rev. A.02.
SGI Altix 4700 – Scaling Performance

- **Small Molecule; Large Basis Set**
  - Even though this was not the explicit target of the improvements, rev. B.01 is faster by 10%-20% when more than 64 cores are used

- **Large Molecule; Small Basis Set**
  - Again, not a direct target, but improvements are even larger. On 64 cores rev. B.01 is now 70%-80% faster with around 50% parallel efficiency

- **Force Calculations**
  - We see again significant improvements in performance for rev. B.01. Good scaling is seen on up to 80 core in the larger case and more than 70% faster on 64 cores

- **Frequency and TD-DFT Jobs**
  - Here we see the largest improvements: on 32 cores rev. B.01 is 2-2.5 times faster and shows efficient parallel scaling on up to 48 cores

- **CIS Calculations**
  - The test used shows similar behavior as the previous cases; better than 2x improvement on 32 cores and efficient parallel scaling on up to 64 cores

- **Post-SCF**
  - Rev. B.01 has the same performance for both the MP2 and MP4 cases as rev. A.02
HF and DFT – Small Molecule and Large Basis Set

**a-pinene HF (678 basis functions)**

- **Jobs/Day**
  - 0
  - 100
  - 200
  - 300
  - 400
  - 500
  - 600
  - 700

- **Number of Cores**
  - 0
  - 20
  - 40
  - 60
  - 80
  - 100
  - 120
  - 140

- **Jobs/Day** vs. **Number of Cores**
  - SGI Altix 4700 9150M 1.66GHz, 24M A.01
  - SGI Altix 4700 9150M 1.66GHz, 24M A.02

**a-pinene DFT (678 basis functions)**

- **Jobs/Day**
  - 0
  - 200
  - 400
  - 600
  - 800
  - 1000
  - 1200
  - 1400

- **Number of Cores**
  - 0
  - 20
  - 40
  - 60
  - 80
  - 100
  - 120
  - 140

- **Jobs/Day** vs. **Number of Cores**
  - SGI Altix 4700 9150M 1.66GHz, 24M B.01
  - SGI Altix 4700 9150M 1.66GHz, 24M A.02
DFT - Large Molecules – Small Basis Sets

**Taxol DFT (1,123 basis functions)**

![Graph showing the relationship between number of cores and jobs/day for Taxol DFT.]

- **Jobs/Day**
  - Scale: 0, 50, 100, 150, 200, 250, 300

- **Number of Cores**
  - Scale: 0, 10, 20, 30, 40, 50, 60, 70

- **SGI Altix 4700 9150M 1.66GHz,24M B.01**
- **SGI Altix 4700 9150M 1.66GHz,24M A.02**

**Ala17 DFT (1,515 basis functions)**

![Graph showing the relationship between number of cores and jobs/day for Ala17 DFT.]

- **Jobs/Day**
  - Scale: 0, 20, 40, 60, 80, 100, 120, 140, 160

- **Number of Cores**
  - Scale: 0, 10, 20, 30, 40, 50, 60, 70

- **SGI Altix 4700 9150M 1.66GHz,24M B.01**
- **SGI Altix 4700 9150M 1.66GHz,24M A.02**
Force Calculations

Test 3997 (Valinomycin) DFT Force

RK397 (Valinomycin) DFT Force (1,620 basis functions)
Frequency and TD-DFT Calculations

**a-pinene Frequency (346 basis functions)**

![Graph showing frequency calculations for a-pinene with different numbers of cores and SGI Altix 4700 9150M 1.66GHz, 24M B.01 and A.02 configurations.]

**TATB TD-DFT/Single&Triple (438 basis functions)**

![Graph showing TATB TD-DFT calculations with different numbers of cores and SGI Altix 4700 9150M 1.66GHz, 24M B.01 and A.02 configurations.]

CESCA Diciembre, 2010

SGI Proprietary
TATB CIS (450 basis functions)

Number of Cores

Jobs/Day

- SGI Altix 4700 9150M 1.66GHz,24M B.01
- SGI Altix 4700 9150M 1.66GHz,24M A.02
Post-SCF – MP2 and MP4

Histidine-H+ complex MP2 FullDirect (591 basis functions)

Histidine-H+ complex MP4 (183 basis functions)

Jobs/Day

Number of Cores

Jobs/Day

Number of Cores

SGI Altix 4700 9150M 1.66GHz, 24M B.01

SGI Altix 4700 9150M 1.66GHz, 24M A.02

SGI Altix 4700 9150M 1.66GHz, 24M B.01

SGI Altix 4700 9150M 1.66GHz, 24M A.02

CESCA Diciembre, 2010

SGI Proprietary
On a single core rev. B.01 is on average slightly faster than rev. A.02 (~4%). This may be a reflection of newer compilers.

Since the “nodes” on SGI Altix UV1000 are larger (12 cores) than those of SGI Altix 4700 (4 cores) the improvements in parallelism in rev. B.01 are also more noticeable (~7%).
SGI Altix UV1000 – Scaling Performance

- Small Molecule; Large Basis Set
  - The most important difference in these cases is the lifting of the 64 core OpenMP limit in the PGI compilers. Now we show parallel scaling with 50% efficiency on around 144 cores
  - Where comparisons is possible rev. B.01 is around 15% faster than rev. A.02

- Large Molecule; Small Basis Set
  - Improvements are similar as on SGI Altix 4700; on 48 cores rev. B.01 is ~50% faster

- Force Calculations
  - We see again significant improvements in performance for rev. B.01. Good scaling is seen on up to 60 core in the larger case and more than 60% faster on 48 cores

- Frequency and TD-DFT Jobs
  - The gains here are more modest than on SGI Altix 4700. Efficient parallel scaling can be seen now at 48 cores. On 24 cores rev. B.01 is ~30% faster

- CIS Calculations
  - The test used shows similar behavior as the previous cases; around 25% improvement on 24 cores and efficient parallel scaling on up to 36 cores

- Post-SCF
  - Rev. B.01 has the same performance for both the MP2 and MP4 cases as rev. A.02
HF and DFT – Small Molecule and Large Basis Set

![Graph showing a-pinene HF (678 basis functions)](image1)

![Graph showing a-pinene DFT (678 basis functions)](image2)
DFT – Large Molecules; Small Basis Sets

**Taxol DFT (1,123 basis functions)**

- SGI Altix UV1000 X7542 2.66GHz, 18M 1067MHz B.01
- SGI Altix UV1000 X7542 2.66GHz, 18M 1067MHz A.02

**Ala17 DFT (1,515 basis functions)**

- SGI Altix UV1000 X7542 2.66GHz, 18M 1067MHz B.01
- SGI Altix UV1000 X7542 2.66GHz, 18M 1067MHz A.02
Force Calculations

Test 3997 (Valinomycin) DFT Force

RK397 (Valinomycin) DFT Force (1,620 basis functions)
Frequency and TD-DFT Calculations

**a-pinene Frequency (346 basis functions)**

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>Jobs/Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

- SGI Altix UV1000 X7542 2.66GHz,18M 1067MHz B.01
- SGI Altix UV1000 X7542 2.66GHz,18M 1067MHz A.02

**TATB TD-DFT/Single&Triple (438 basis functions)**

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>Jobs/Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

- SGI Altix UV1000 X7542 2.66GHz,18M 1067MHz B.01
- SGI Altix UV1000 X7542 2.66GHz,18M 1067MHz A.02
TATB CIS (450 basis functions)

- **Jobs/Day**
  - 0
  - 100
  - 200
  - 300
  - 400
  - 500
  - 600
  - 700

- **Number of Cores**
  - 0
  - 10
  - 20
  - 30
  - 40

**Legend**:
- Red: SGI Altix UV1000 X7542 2.66GHz, 18M 1067MHz B.01
- Blue: SGI Altix UV1000 X7542 2.66GHz, 18M 1067MHz A.02
Post-SCF – MP2 and MP4

Histidine-H+ complex MP2 FullDirect (591 basis functions)

Jobs/Day

Number of Cores

Histidine-H+ complex MP4 (183 basis functions)

Jobs/Day

Number of Cores

BSI Altix UV1000 X7542 2.66GHz, 18M 1067MHz A.01

BSI Altix UV1000 X7542 2.66GHz, 18M 1067MHz A.02
Summary

- Gaussian 09 rev. B.01 has been released by Gaussian, Inc. on August 12, 2010. This revision includes, among others, important improvements in shared-memory parallelism and the lifting of the limit of 64 cores for OpenMP parallelism in the x86_64 binaries.
- This report compares the performance of rev. B.01 with the previous revision, A.02, on 2 SGI systems: two shared-memory machines (SGI Altix 4700 and SGI Altix UV1000).
- The improvements on SGI Altix 4700 are the most noticeable since this was the original target.
  - In general good parallel efficiency has been extended to higher number of cores
  - Frequency calculations and TD-DFT jobs can be more than 2x faster on 32 cores
- There are two relevant improvements for SGI Altix UV1000:
  - Compiler limits to OpenMP scaling have been lifted, we see now efficient parallel scaling on up to 144 cores
  - The shared-memory lock fix can improve performance on moderate core-counts (40-60 cores) by as much as 60%
- In the two machines considered there was no improvement in the Post-SCF jobs