Application Performance: Altix vs. the Rest

Georg Hager

Computing Center, University of Erlangen (RRZE)

SGI User Group Conference 2004

Agenda

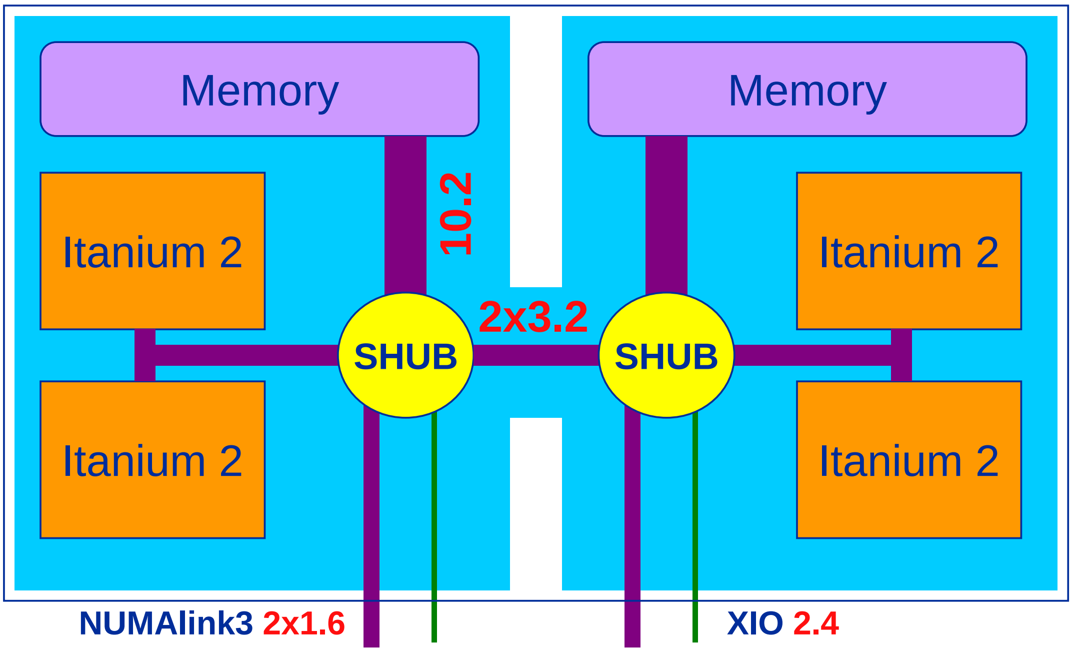
- RRZE
- Altix Basics
  - Architecture
  - Software
  - Competitors
- Applications
  - CFD
    - SIP Solver (OpenMP)
  - Physics
    - DMRG (SCSL)
    - DMRG (OpenMP)
- Conclusions

RRZE Machine Structure

- Fujitsu VPP300
  - 8 CPUs @ 2.2 GFlops, 2 GB
  - Installed 1997
- SGI Origin 3400
  - 28 CPUs @ 500 MHz (R14k)
  - 56 GB
  - Installed June 2001
- IA32 Cluster
  - 86x2 Xeon 2.66 GHz
  - Gigabit Ethernet
  - Installed April 2003
- SGI Altix 3700
  - 28 CPUs @ 1.3 GHz
  - 112 GB
  - Installed December 2003
- Several test systems
  - Opteron, Xeon, IT1, IT2

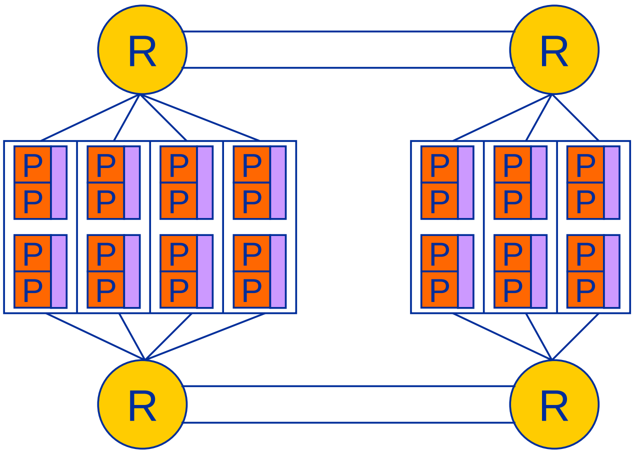
Altix Basics

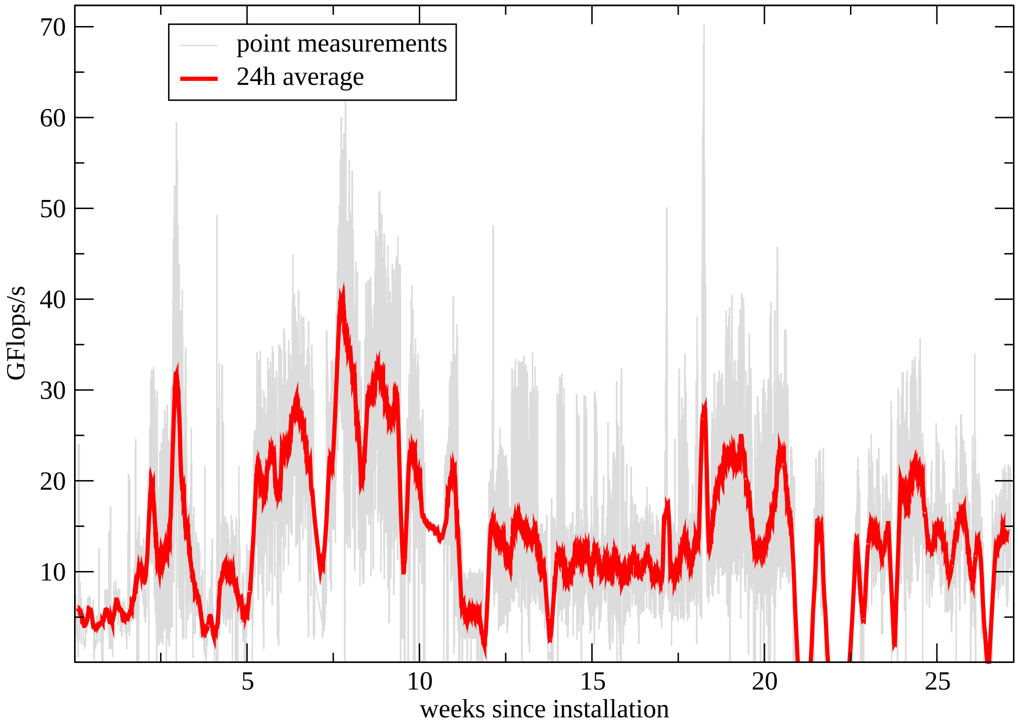
- Dual-Itanium2 nodes



Altix Basics

- **ccNUMA via NumaLink3/4**
  - **Layout of Altix at RRZE:**





Altix Software

- OS: Linux w/ ProPack 2.4
- Profiling tools
  - pfmon
  - profile.pl
  - histx (cool, but no match for Speedshop)
- Compilers
  - Intel 7.1 and 8.0, F90 and C++
  - gcc 3.0.4

Altix Competitors

- IBM p690
  - 32-CPU ccNUMA
  - dual Core Power4
- NEC TX7
  - 32-CPU ccNUMA, Itanium 2
  - 4 CPUs per memory path
- NEC SX6
  - Vector CPU, 0.5 words/flop
  - Shared memory node with 8 CPUs, full bandwidth
- Intel Xeon systems
  - IA32 architecture, high clock rates
- AMD Opteron systems
  - X86-64 architecture, enhancements over IA32
  - one path to memory per CPU
  - 8-CPU SMPs with hardly any external hardware

CFD: Strongly Implicit Solver (SIP)

- CFD: Solving **A x = b**
  for finite volume methods can be done by Strongly-Implicit-Procedure (SIP) according to Stone
- SIP-solver is widely used:
  - LESOCC, FASTEST, FLOWSI **(Institute of Fluid Mechanics, Erlangen)**
  - STHAMAS3D **(Crystal Growth Laboratory, Erlangen)**
  - CADiP **(Theoretical Thermodynamics and Transport Processes, Bayreuth)**
  - …
- SIP-Solver: 1) Incomplete LU-factorization
  2) Series of forward/backward substitutions
- Toy program available at: ftp.springer.de in /pub/technik/peric (M. Peric)

SIP-solver
Data dependencies & Implementations

Basic data dependency: **(i,j,k)←{(i-1,j,k);(i,j-1,k);(i,j,k-1)}**

```
do k = 2 , kMax
   do j = 2 , jMax
      do i = 2 , iMax
           RES(i,j,k) ={RES(i,j,k)-LB(i,j,k)*RES(i,j,k-1)
$         - LW(i,j,k)*RES(i-1,j,k)- LS(i,j,k)*RES(i,j-1,k)
$         }*LP(i,j,k)
      enddo
   enddo
enddo
```

3-fold nested loop (3D): **(i,j,k)**

- Data locality (Caches !)
- No shared memory parallelization (Hitachi: *Pipeline parallel processing*)

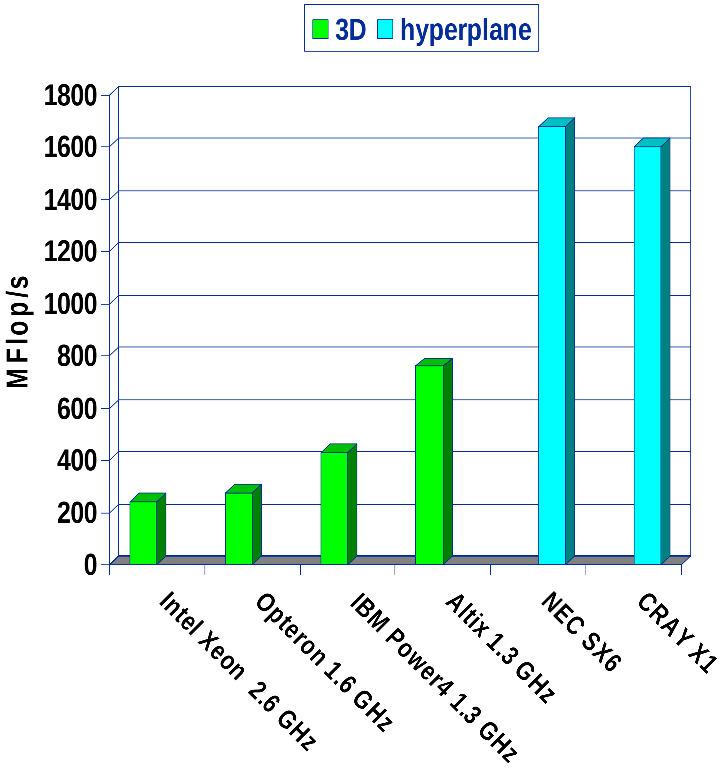
Hyperplane: **(i+j+k=const)**

- Non-contiguous memory access
- vectorization of innermost loop
- unsuitable for RISC systems

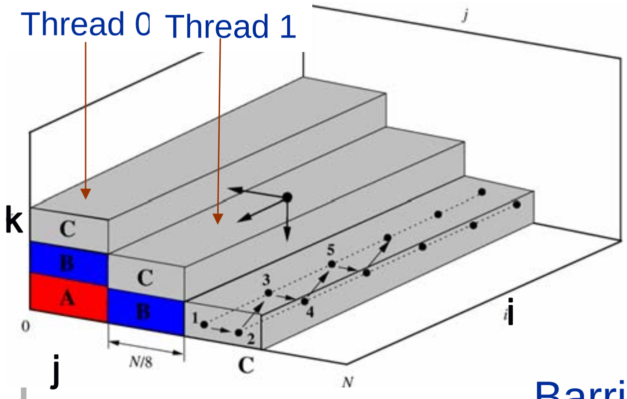
SIP-solver
Basic Performance Numbers

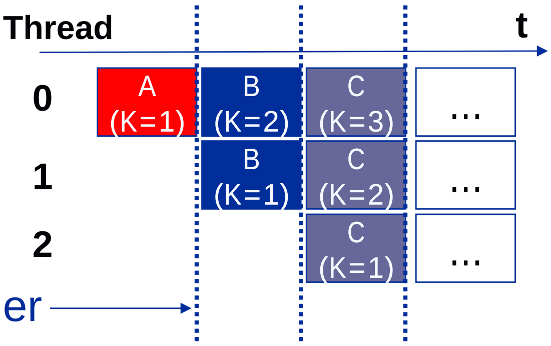
- Benchmark
  - Lattice: 91^3
  - 100 MB
  - 1 ILU
  - 500 iterations

- X1: 1 MSP
- SX6: 1 CPU



SIP-solver
Pipeline Parallel Processing





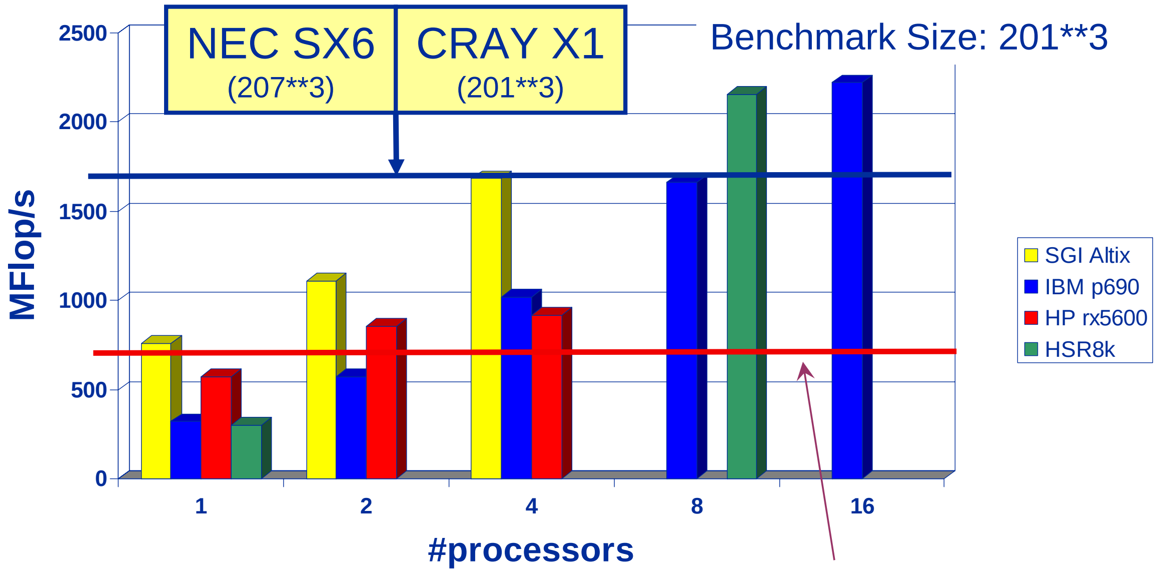
Thread				
0	A (K=1)	B (K=2)	C (K=3)	…
1		B (K=1)	C (K=2)	…
2			C (K=1)	…

- Split up j-loop in chunks of equal size for each thread
- Split up k-direction in blocks of equal chunks
- Pipelining in k-direction: Only one k-Index is active at a fixed time
- Efficient parallelisation if **`kMax, jMax >> #Threads`**

SIP-solver
Pipeline Parallel Processing using OpenMP

```
$omp parallel private(….)

do l =2, kMax+numThreads-2,1

   threadID=OMP_GET_THREAD_NUM()
   k = l – threadID

   if((k.ge.2).and.(k.le.kMaxM)) then

      do j = jS(threadID),jE(threadID)
         do i = 2 , iMax
            RES(i,j,k) = {RES(i,j,k)-

$              LB(i,j,k)*RES(i,j,k-1) …  ! Same thread

$       …      LS(i,j,k)*RES(i,j-1,k) …} ! threadID-1 in
         enddo                            ! in prev. l-iter
      enddo
   endif
$omp barrier
enddo
$omp end parallel
```

CFD kernel: SIP-solver
SMP Performance vs. Vector CPUs

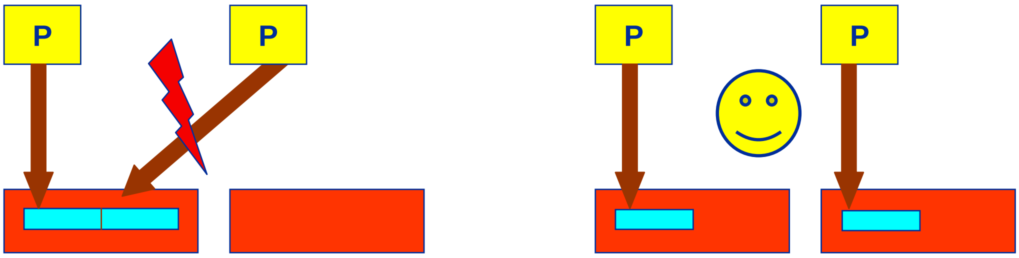


- IBM p690: speedup(16)=7
- Hitachi SR8000: speedup(8)=7.2
- HP rx5600: Intel Itanium2 – 1 GHz; one memory path!

Altix 4-CPU starting point

SIPSolver (ppp):
NUMA Placement Problems

- Internode communication on Altix is significantly slower than intra-node (factor of 2 worse than Origin)
- Consequence: Data locality is even more important than on Origin
  - "First Touch" policy maps memory pages in the node where they are first used
  - Initialization of data structures must be parallelized to ensure proper placement!



SIPSolver (ppp):
NUMA Placement Problems

- Change parallelization of initialization loop:

```
!$omp parallel do private(i,j)
do k=1,kMax
   do j=1,jMax
      do i=1,iMax
         T(i,j,k)=0.
      enddo
   enddo
enddo
```

```
do k=1,kMax
!$omp parallel do private(i)
   do j=1,jMax
      do i=1,iMax
         T(i,j,k)=0.
      enddo
   enddo
enddo
```

DMRG

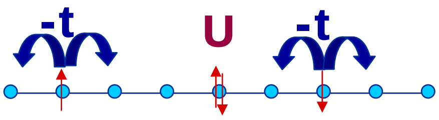
- Density-Matrix Renormalization Group
- Used in solid state physics and quantum chemistry to explore properties of low-lying states in quantum systems
- Competing methods: ED (Exact Diagonalization) and QMC (Quantum Monte Carlo)
- Especially compared to ED, many problems can be tackled on workstations instead of supercomputers

- Goal: Parallelize / port existing C++ DMRG code to modern shared-memory systems
- Use the parallelized code to push manageable physical system sizes (# of sites) to new heights

Motivation – Microscopic Models

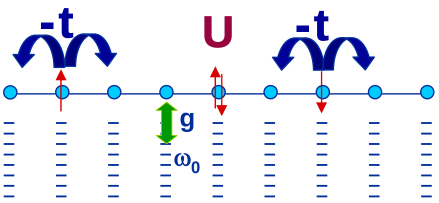
- Microscopic Hamiltonians in second quantization
e.g. Hubbard model

$$H = -t \sum_{\langle ij \rangle, \sigma} \left[c_{i\sigma}^{\dagger} c_{j\sigma} + \mathrm{H.c.} \right] + U \sum_{i} n_{i\uparrow} n_{i\downarrow}$$



e.g. Holstein-Hubbard model (HHM)

$$H = -t \sum_{\langle ij \rangle, \sigma} \left[c_{i\sigma}^{\dagger} c_{j\sigma} + \mathrm{H.c.} \right] + U \sum_{i} n_{i\uparrow} n_{i\downarrow} + g\omega_0 \sum_{i,\sigma} (b_i^{\dagger} + b_i) n_{i\sigma} + \omega_0 \sum_{i} b_i^{\dagger} b_i$$



Hilbert space / #quantum states growth exponentially

HHM using an N-site lattice: 4^N * $(M+1)^N$ (N~10-100; M~10)

Electrons Phonons: Max. M per Site

Motivation – Numerical Approaches to Ground-State Properties

Traditional Approaches

- **Quantum Monte Carlo (QMC)**
- **Exact Diagonalizaton (ED):** Massively Parallel Codes on Supercomputers

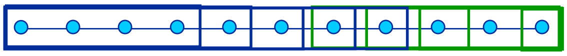
New Approach

- **Density Matrix Renormalization Group (DMRG) Method**
  - **Originally introduced by White in 1992**
  - **Large sequential C++ package is in wide use (quantum physics and quantum chemistry)**
  - **Elapsed Times: hours to weeks with desktop CPUs**
  - **No parallel implementation available to date**

DMRG Algorithm

DMRG algorithm (finite size; left to right sweep)

1. Diagonalize reduced DM for a system block of size *l* and extract *m* eigenvectors with largest eigenvalue
2. Construct all relevant operators (system block & environment,…) for a system block of size *l+1* in the reduced density matrix eigenbasis
3. Form a superblock Hamiltonian from system & environment Hamiltonians plus two single sites
4. **Diagonalize new superblock Hamiltonian**



Accuracy depends mainly on *m* (*m*~100 – 10000)

DMRG Algorithm

Implementation

- Start-Up with infinite-size algorithm
- DM diagonalization: LAPACK (dsyev) costs about 5 %
- **Superblock diagonalization costs about 90 % (Davidson algorithm)**
- **Most time-consuming step: Sparse matrix-vector multiply (MVM) in Davidson (costs about 85 %)**
- **Sparse matrix *H* is constructed by the transformations of each operator in *H*:**

$$H_{ij;i'j'} = \sum_{\alpha} A^{\alpha}_{ii'} B^{\alpha}_{jj'}$$

Contribution from system block and from environment

DMRG Algorithm: Parallelization

Implementation of sparse MVM

- **Sparse MVM: Sum over dense matrix-matrix multiplies!**

$$\sum_{i'j'} H_{ij;i'j'} \psi_{i'j'} = \sum_{\alpha} \sum_{i'} A^{\alpha}_{ii'} \sum_{j'} B^{\alpha}_{jj'} \psi_{i'j'}$$

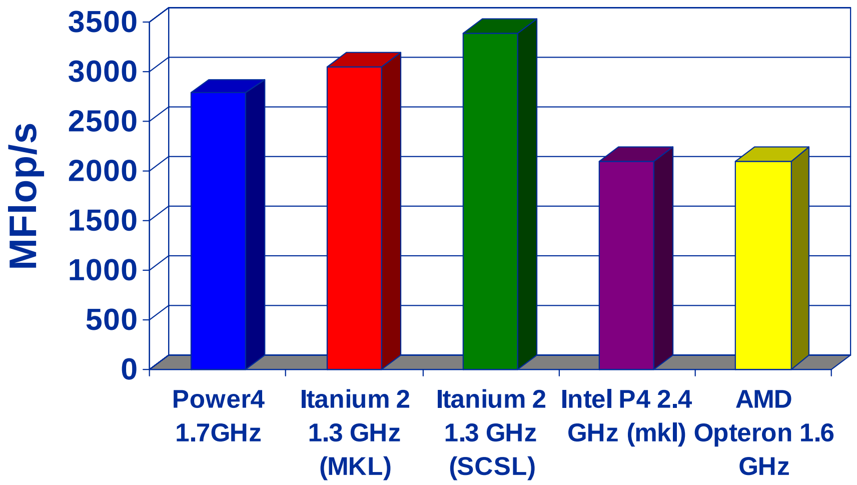
- **However *A* and *B* may contain only a few nonzero elements, e.g. if conservation laws (quantum numbers) have to be obeyed**
- **To minimize overhead an additional loop (running over nonzero blocks only) is introduced**

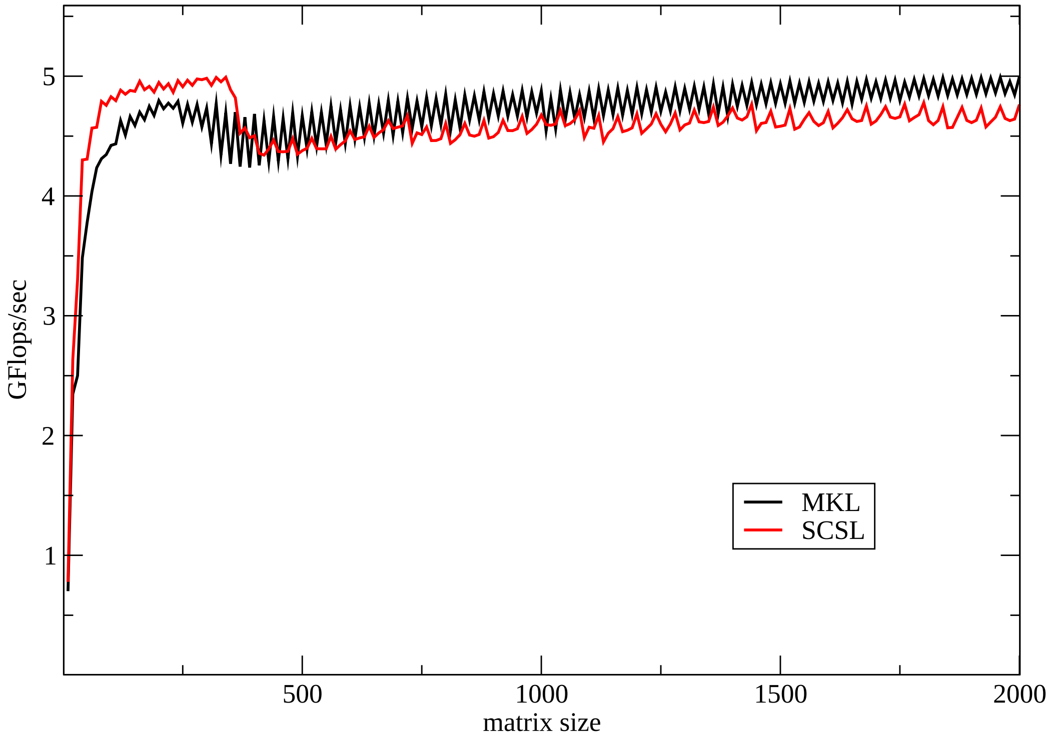
$$H\psi = \sum_{\alpha} \sum_{k} (H\psi)^{\alpha}_{L(k)} = \sum_{\alpha} \sum_{k} A^{\alpha}_{k} \psi_{R(k)} \left[B^{\mathrm{T}}\right]^{\alpha}_{k}$$

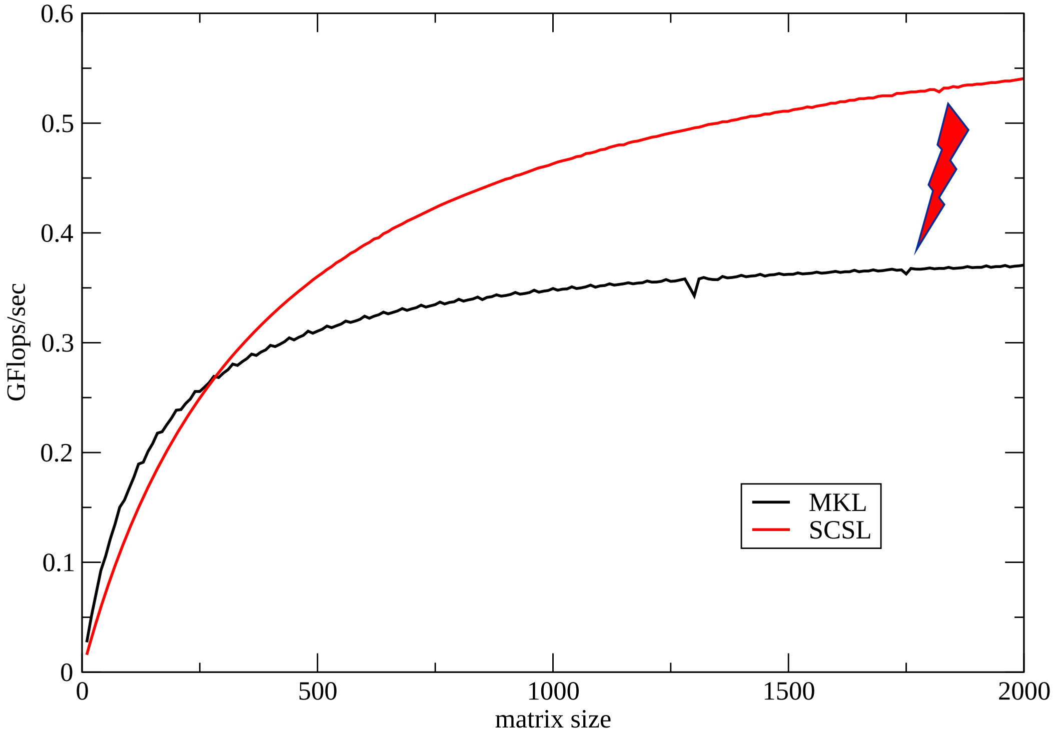
- **Dense matrix-matrix multiplies are implemented using DGEMM from BLAS**

DMRG Serial Performance

- **DGEMM core leads to a significant fraction of peak for real-life problems**
- **Itanium 2 is competitive with respect to Power4**
- **Choice on Altix: Use MKL or SCSL?**
- **DMRG performs better with SCSL!**







DMRG: Potential Parallelization approaches

1. **Linking with parallel BLAS (DGEMM)**
   - Does not require restructuring of code
   - Significant speedup only for large (transformation) matrices (A , B)

2. **Shared-Memory parallelization of outer loops**
   - Chose OpenMP for portability reasons
   - Requires some restructuring & directives
   - Speedup should not depend on size of (transformation) matrices

   Maximum speedup for total program:
   - if MVM (accounts for 85%) is parallelized only: ~6 - 8

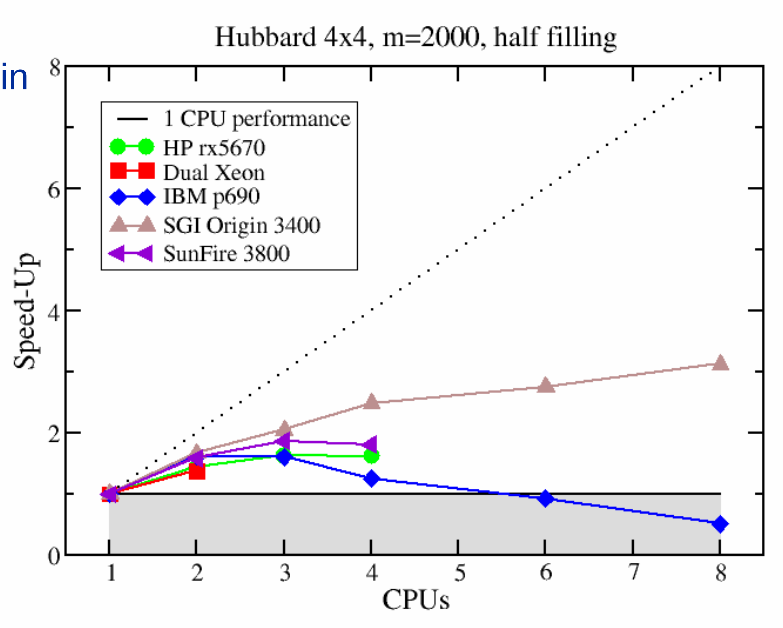
MPI parallelization

- Requires complete restructuring of algorithm -> new code

DMRG: Parallel BLAS

Linking with parallel BLAS

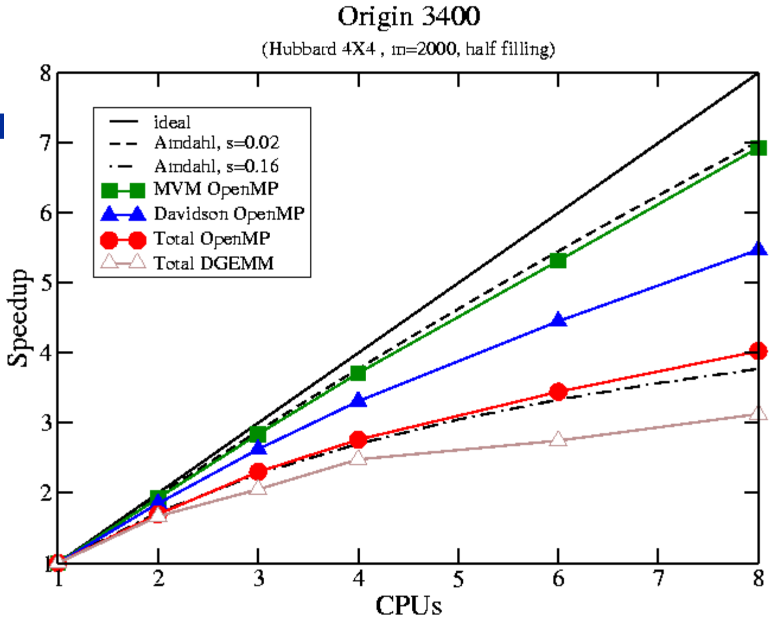
- Useless on IBM for #CPU > 4
- Best scalability on Origin (Network, BLAS implementation)
- Dual processor nodes can reduce elapsed runtime by about 30 %
- Increasing *m* to 7000: S(4) = 3,2 (SGI & HP)
- Small *m* (~600) with HHM: No Speed-Up



DMRG: OpenMP Parallelization

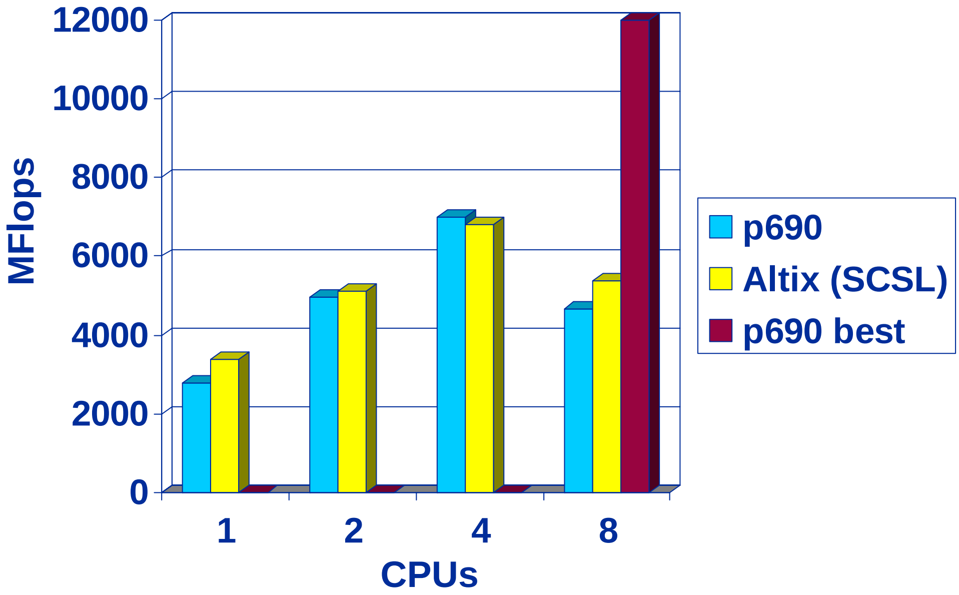
Scalability on Origin

- **OpenMP scales significantly better than parallel DGEMM**
- **Serial overhead in parallel MVM is only about 2%!**
- **Linking with parallel BLAS gives an additional performance gain of 15 %!**



DMRG Scalability

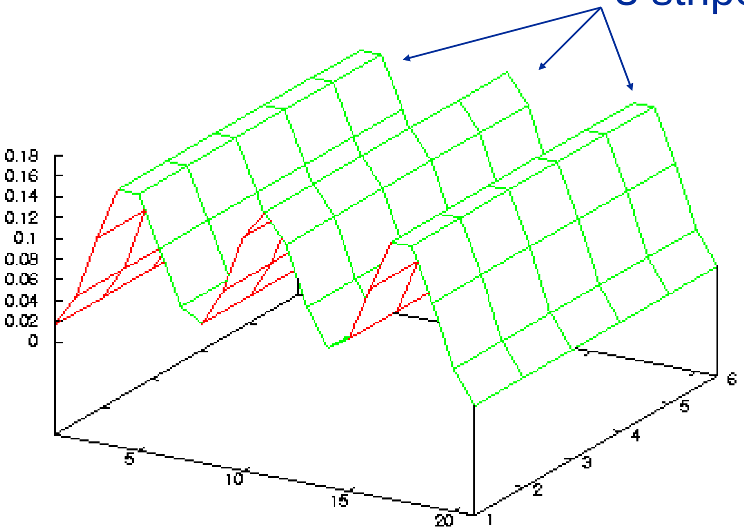
- Comparison of DMRG scalability



Application: Ground State of 21x6 (OpenxPeriodic) BCs Hubbard ladder with 12 holes

Hole density at m=6000, U=12

3 stripes!



Previously possible only with up to 7x6 sites!

Parallel DMRG: References

- G. Hager, E. Jeckelmann, H. Fehske, and G. Wellein: *Parallelization Strategies for Density Matrix Renormalization Group Algorithms on Shared-Memory Systems.*
  J. Comp. Phys. 194, 795 (2004)

- G. Hager, E. Jeckelmann, H. Fehske, and G. Wellein:
  Exact Numerical Treatment of Finite Quantum Systems using Leading-Edge Supercomputers.
  To be published in: *Proceedings of the International Conference on High Performance Scientific Computing*, March 10-14 2003, Hanoi, Vietnam (Springer)

- G. Hager, E. Jeckelmann, H. Fehske, and G. Wellein:
  Investigation of Stripe Formation in Hubbard Ladders using parallel DMRG
  To be published in: *High Performance Computing in Science and Engineering Munich 2004*, March 2-3 2004, Munich, Germany (Springer)

Conclusions

- **Memory placement is very important on Altix**
  - **initialize data structures the way they are accessed later**
  - **does not seem to be that much of a problem with p690**

- **SCSL is worth a look as an alternative to MKL**
  - **MKL is getting better**
  - **"nested parallelism" issues with MKL**

- **OpenMP scalability problems on Altix**
  - **Hope for compiler improvements?**
  - **p690 suffers from similar symptoms, but the cause is different**

- **Altix outperforms p690 for CFD Codes**