ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

Towards a hybrid multi-core implementation of MAQUIS Exact Diagonalization (ED)

Sergei Isakov, Will Sawyer, Adrian Tineo, Gilles Fourestey, Neil Stringfellow, Matthias Troyer

Overarching goals of our group's work

Overarching goals of our group's work

Use *scientifically relevant* mini-apps to:

Overarching goals of our group's work

Use *scientifically relevant* mini-apps to:

- Evaluate emerging architectures
  - AMD Interlagos
  - Intel Sandybridge
  - IBM BG/Q, GPUs, if possible

Overarching goals of our group's work

Use *scientifically relevant* mini-apps to:

- Evaluate emerging architectures
  - AMD Interlagos
  - Intel Sandybridge
  - IBM BG/Q, GPUs, if possible
- Evaluate programming paradigms
  - MPI + OpenMP hybrid programming
  - MPI-2 one-sided communication
  - SHMEM
  - UPC (as implemented in Cray compiler)
  - OpenACC, if possible

Overarching goals of our group's work

Use *scientifically relevant* mini-apps to:

- Evaluate emerging architectures
  - AMD Interlagos
  - Intel Sandybridge
  - IBM BG/Q, GPUs, if possible
- Evaluate programming paradigms
  - MPI + OpenMP hybrid programming
  - MPI-2 one-sided communication
  - SHMEM
  - UPC (as implemented in Cray compiler)
  - OpenACC, if possible
- Compare performance across platforms
  - out-of-the-box performance
  - evaluate optimization effort
  - socket-for-socket, node-for-node comparisons

CSCS Testbed Platforms

System name	Rivera	Sandy	Castor
Processor	AMD 6274	Intel E5-2680	Intel X5650
Proc. nickname	Interlagos	Sandybridge	Westmere
Clock (GHz)	2.2	2.7	2.66
Sockets/Node	2	2	2
Cores/Socket	16	8	6
NUMA/Socket	2	1	1
DP GFlops/Socket	140.8	172.8	63.8
Memory/Socket	16 GB	16 GB	12 GB
DDR3 mem. speed	1600	1333	1333
L1 cache (excl.)	16KB	32KB	32KB
L2 cache/# cores	2MB/2	256KB/1	256KB/1
L3 cache/# cores	8MB/8	20MB/8	12MB/6
Hyperthreading?	no	yes (2)	unenabled
TPA/Socket (W)	115	130	95

Parallel Test Platforms

System name	Rosa	Todi	Rothorn	Grotius
Product name	Cray XE6	Cray XT6	SGI UV1000	IBM BG/Q
Interconnection	Gemini	Gemini	NUMAlink	5D Torus
Processor	AMD 6272	AMD 6272	Intel E7-8837	IBM A2
Proc. nickname	Interlagos	Interlagos	Westmere	
Clock (GHz)	2.1	2.1	2.66	1.60
Sockets/Node	2	1	32	1
Cores/Socket	16	16	8	16
NUMA/Socket	2	2	1	1
DP GFlops/Socket	134.4	134.4	85.1	204.8
Memory/Socket	16 GB	16 GB	64 GB	16 GB
DDR3 mem. speed	1600	1600	1333	1333
L1 cache (excl.)	16KB	16KB	32KB	32KB
L2 cache/# cores	2MB/2	2MB/2	256KB/1	32MB/16
L3 cache/# cores	8MB/8	8MB/8	24MB/8	N/A
Hyperthreading?	no	no	no	yes (4)
TPA/Socket (W)	115	115	130	60

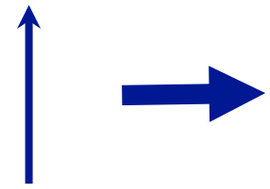
Fundamental ED problem

$$H = J \sum S_i^z S_j^z + \Gamma \sum S_i^x$$

Fundamental ED problem

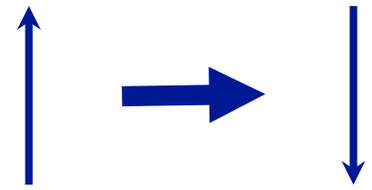
$$H = J \sum S_i^z S_j^z + \Gamma \sum S_i^x$$

Fundamental ED problem



$$H = J \sum S_i^z S_j^z + \Gamma \sum S_i^x$$

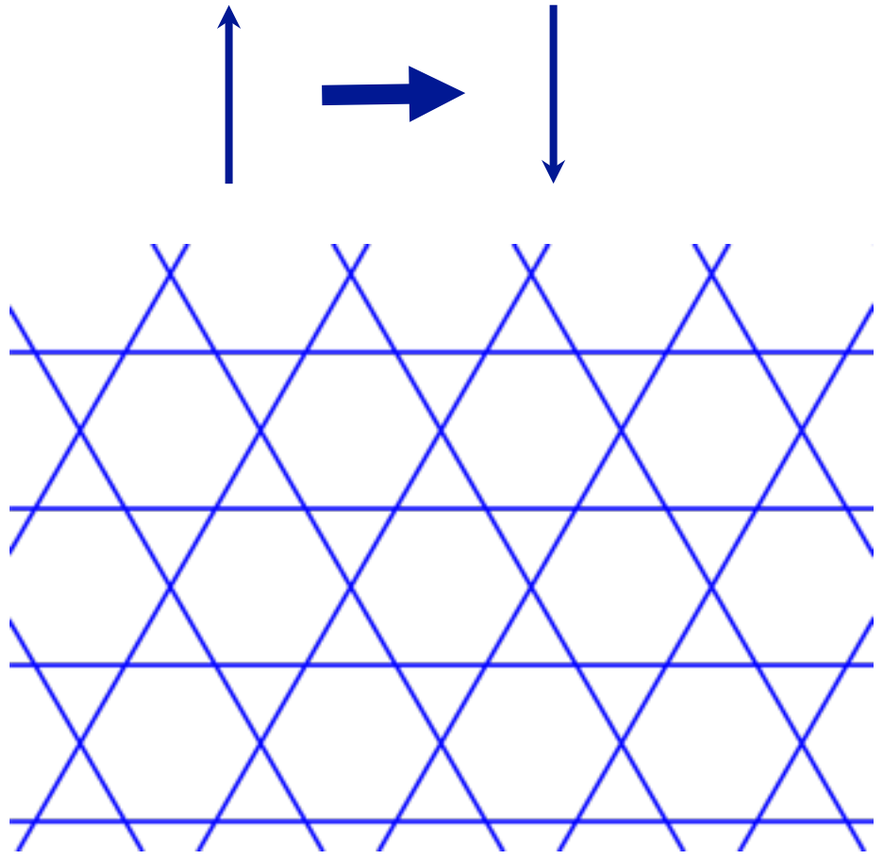
Fundamental ED problem



$$H = J \sum S_i^z S_j^z + \Gamma \sum S_i^x$$

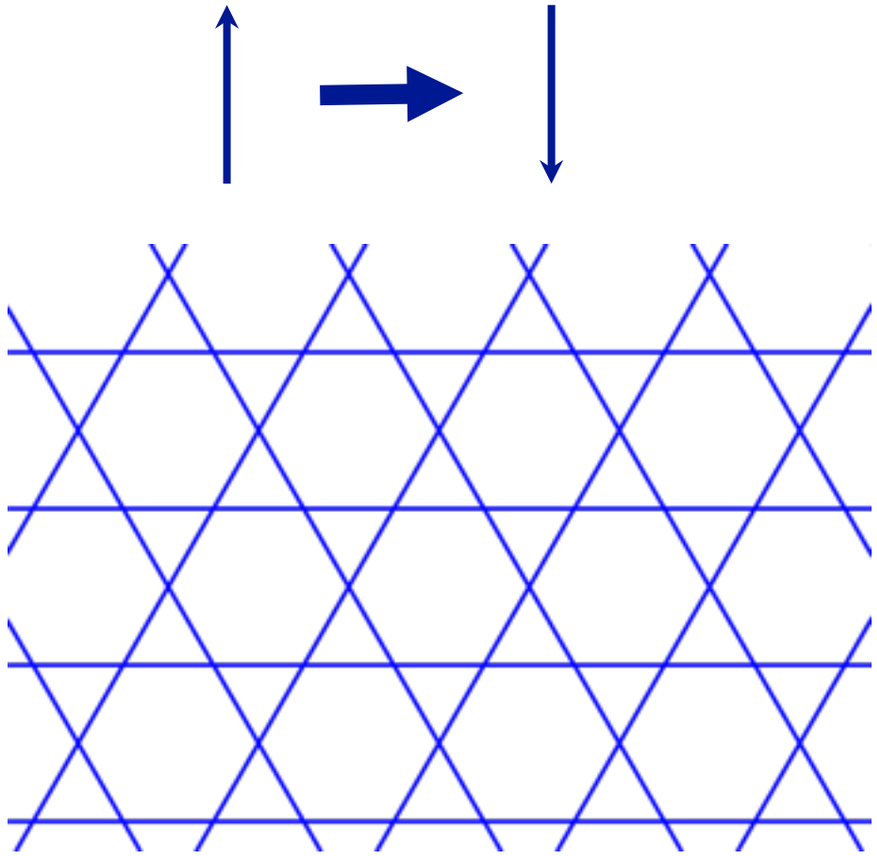
Fundamental ED problem

$$H = J \sum S_i^z S_j^z + \Gamma \sum S_i^x$$



Fundamental ED problem

$$H = J \sum S_i^z S_j^z + \Gamma \sum S_i^x$$

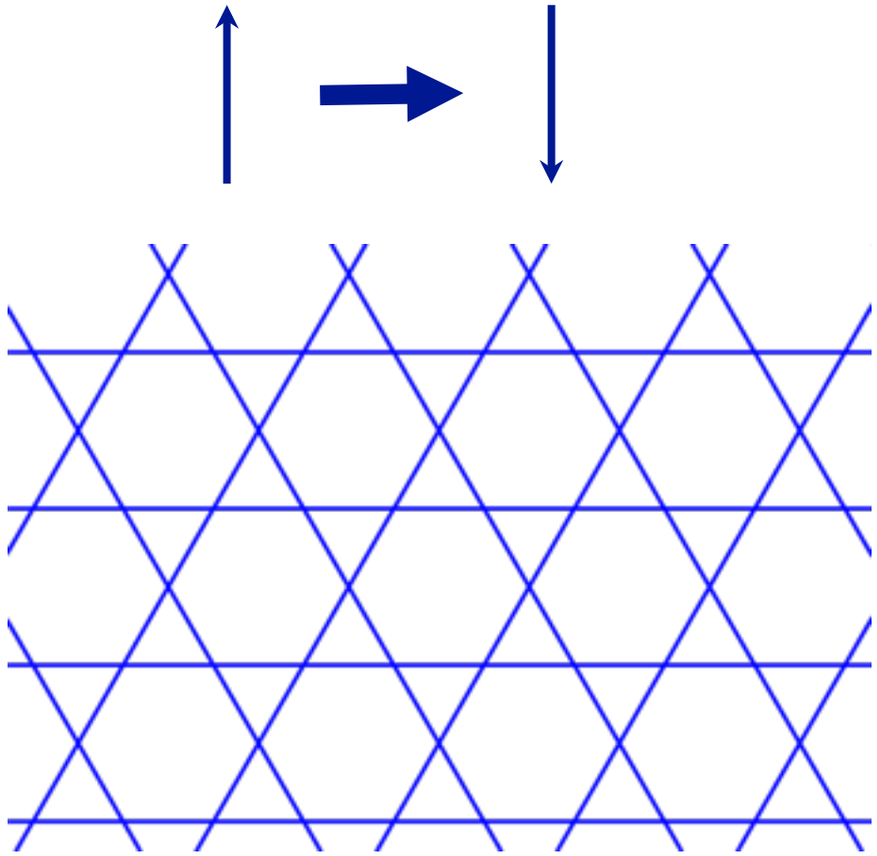


Algorithm 1 *Lanczos iteration*

1: $v_1 \leftarrow$ random vector with norm 1
2: $w_0 \leftarrow 0$
3: $\beta_1 \leftarrow 0$
4: **for** $j = 1, \ldots, r$ **do**
5: $\quad w_j \leftarrow Hv_j - \beta_j v_{j-1}$
6: $\quad \alpha_j \leftarrow (w_j, v_j)$
7: $\quad \beta_{j+1} \leftarrow \|w_j\|$
8: $\quad v_{j+1} \leftarrow w_j / \beta_{j+1}$
9: **end for**

Fundamental ED problem

$$H = J \sum S_i^z S_j^z + \Gamma \sum S_i^x$$



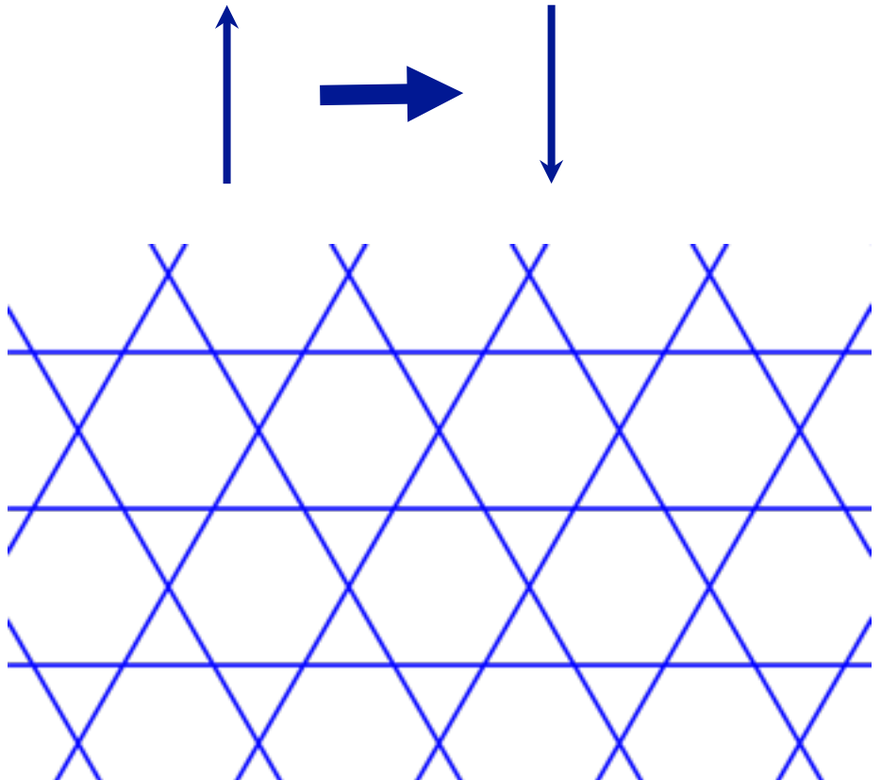
Algorithm 1 *Lanczos iteration*

1: $v_1 \leftarrow$ random vector with norm 1
2: $w_0 \leftarrow 0$
3: $\beta_1 \leftarrow 0$
4: **for** $j = 1, \ldots, r$ **do**
5: $\quad w_j \leftarrow Hv_j - \beta_j v_{j-1}$
6: $\quad \alpha_j \leftarrow (w_j, v_j)$
7: $\quad \beta_{j+1} \leftarrow \|w_j\|$
8: $\quad v_{j+1} \leftarrow w_j / \beta_{j+1}$
9: **end for**

Fundamental ED problem

$$H = J \sum S_i^z S_j^z + \Gamma \sum S_i^x$$

- Any lattice with n sites, 2^n states



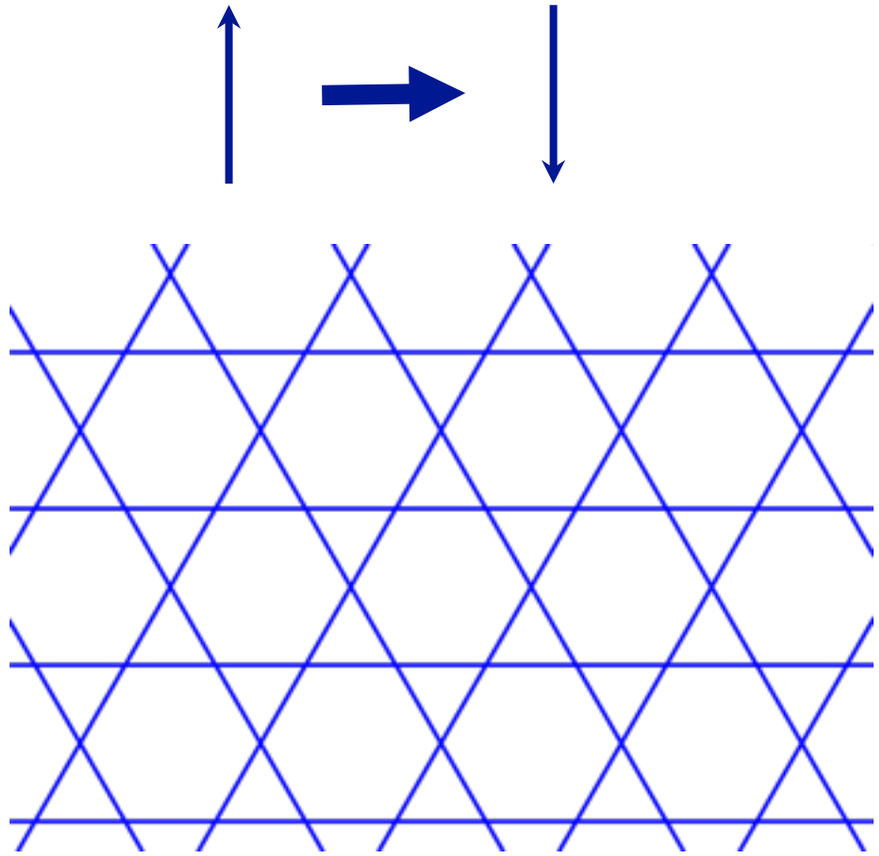
Algorithm 1 *Lanczos iteration*

1: $v_1 \leftarrow$ random vector with norm 1
2: $v_0 \leftarrow 0$
3: $\beta_1 \leftarrow 0$
4: **for** $j = 1, \ldots, r$ **do**
5: $\quad w_j \leftarrow Hv_j - \beta_j v_{j-1}$
6: $\quad \alpha_j \leftarrow (w_j, v_j)$
7: $\quad \beta_{j+1} \leftarrow \|w_j\|$
8: $\quad v_{j+1} \leftarrow w_j / \beta_{j+1}$
9: **end for**

Fundamental ED problem

$$H = J \sum S_i^z S_j^z + \Gamma \sum S_i^x$$

- Any lattice with n sites, 2^n states
- Lanczos eigensolver



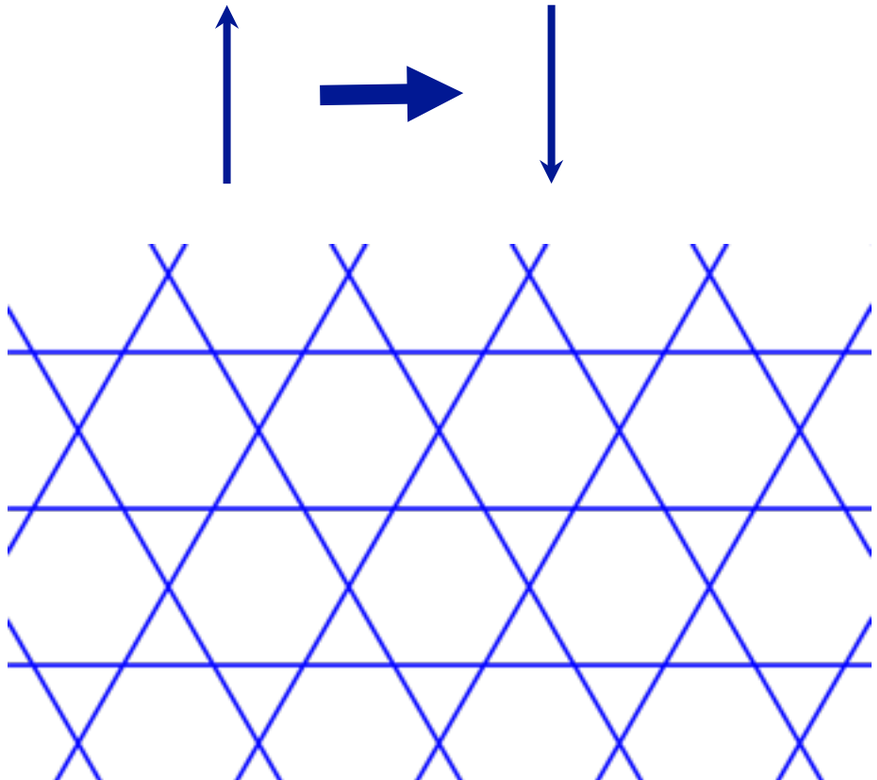
Algorithm 1 *Lanczos iteration*

1: $v_1 \leftarrow$ random vector with norm 1
2: $v_0 \leftarrow 0$
3: $\beta_1 \leftarrow 0$
4: **for** $j = 1, \ldots, r$ **do**
5: $\quad w_j \leftarrow Hv_j - \beta_j v_{j-1}$
6: $\quad \alpha_j \leftarrow (w_j, v_j)$
7: $\quad \beta_{j+1} \leftarrow \|w_j\|$
8: $\quad v_{j+1} \leftarrow w_j / \beta_{j+1}$
9: **end for**

Fundamental ED problem

$$H = J \sum S_i^z S_j^z + \Gamma \sum S_i^x$$

- Any lattice with n sites, 2^n states
- Lanczos eigensolver
- Large, sparse symmetric mat-vec



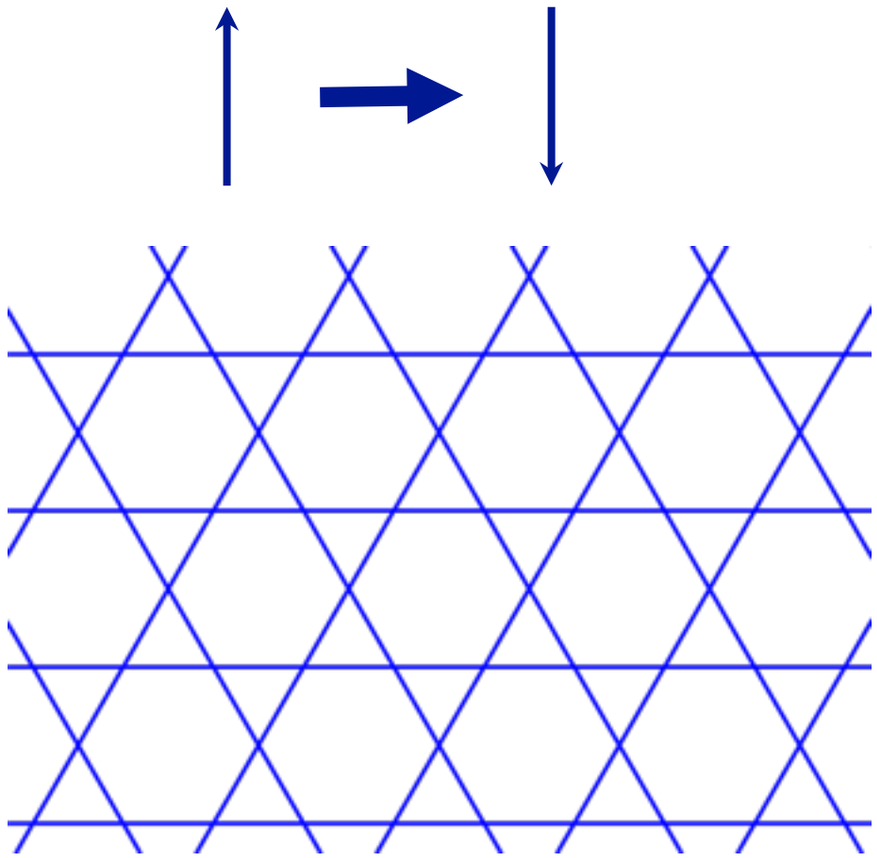
Algorithm 1 *Lanczos iteration*

1: $v_1 \leftarrow$ random vector with norm 1
2: $v_0 \leftarrow 0$
3: $\beta_1 \leftarrow 0$
4: **for** $j = 1, \ldots, r$ **do**
5: $\quad w_j \leftarrow Hv_j - \beta_j v_{j-1}$
6: $\quad \alpha_j \leftarrow (w_j, v_j)$
7: $\quad \beta_{j+1} \leftarrow \|w_j\|$
8: $\quad v_{j+1} \leftarrow w_j / \beta_{j+1}$
9: **end for**

Fundamental ED problem

$$H = J \sum S_i^z S_j^z + \Gamma \sum S_i^x$$

- Any lattice with n sites, 2^n states
- Lanczos eigensolver
- Large, sparse symmetric mat-vec
- Operator has integer operations



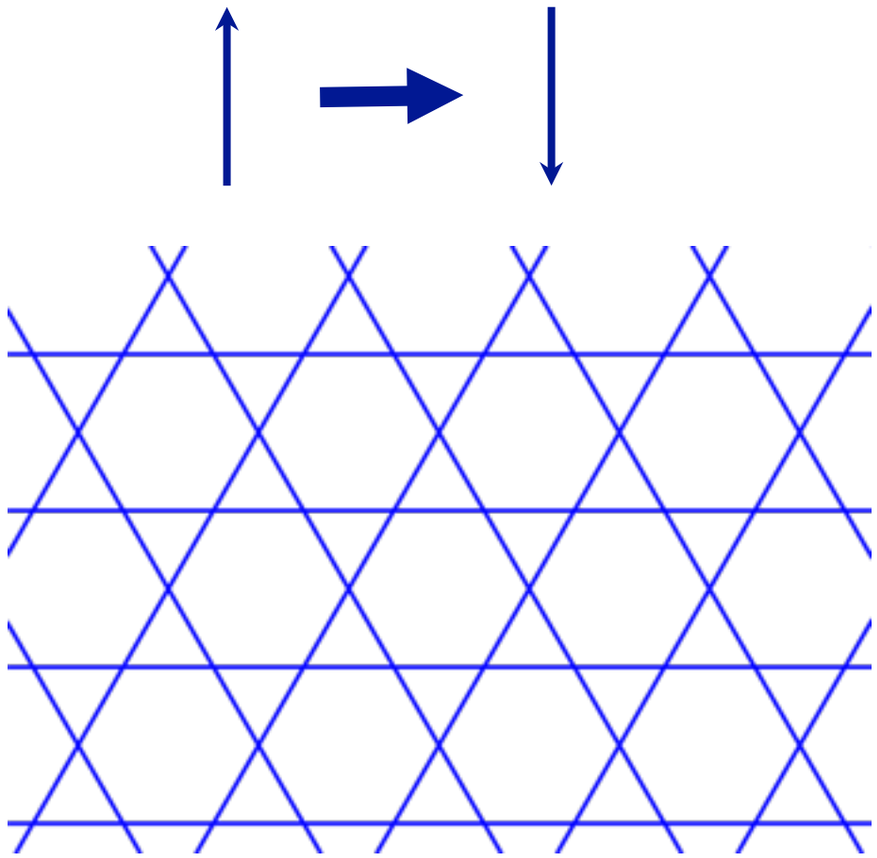
Algorithm 1 *Lanczos iteration*

1: $v_1 \leftarrow$ random vector with norm 1
2: $v_0 \leftarrow 0$
3: $\beta_1 \leftarrow 0$
4: **for** $j = 1, \ldots, r$ **do**
5: $\quad w_j \leftarrow Hv_j - \beta_j v_{j-1}$
6: $\quad \alpha_j \leftarrow (w_j, v_j)$
7: $\quad \beta_{j+1} \leftarrow \|w_j\|$
8: $\quad v_{j+1} \leftarrow w_j / \beta_{j+1}$
9: **end for**

Fundamental ED problem

$$H = J \sum S_i^z S_j^z + \Gamma \sum S_i^x$$

- Any lattice with n sites, 2^n states
- Lanczos eigensolver
- Large, sparse symmetric mat-vec
- Operator has integer operations
- Very irregular sparsity, but



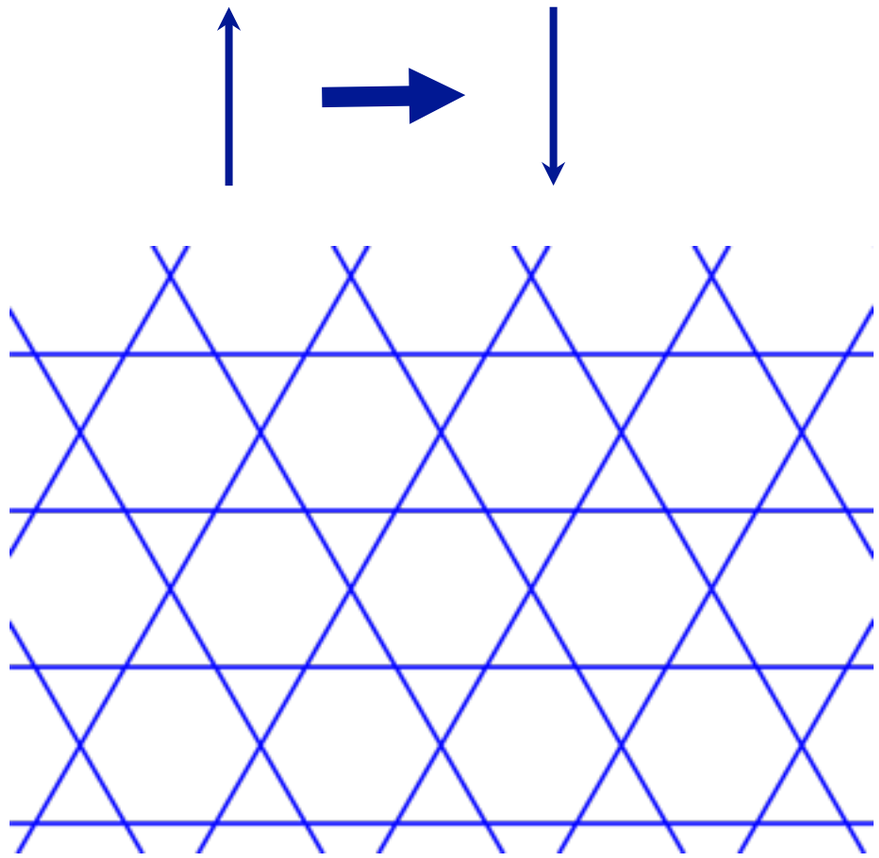
Algorithm 1 *Lanczos iteration*

1: $v_1 \leftarrow$ random vector with norm 1
2: $v_0 \leftarrow 0$
3: $\beta_1 \leftarrow 0$
4: **for** $j = 1, \ldots, r$ **do**
5: $\quad w_j \leftarrow Hv_j - \beta_j v_{j-1}$
6: $\quad \alpha_j \leftarrow (w_j, v_j)$
7: $\quad \beta_{j+1} \leftarrow \|w_j\|$
8: $\quad v_{j+1} \leftarrow w_j / \beta_{j+1}$
9: **end for**

Fundamental ED problem

$$H = J \sum S_i^z S_j^z + \Gamma \sum S_i^x$$

- Any lattice with n sites, 2^n states
- Lanczos eigensolver
- Large, sparse symmetric mat-vec
- Operator has integer operations
- Very irregular sparsity, but
- Limited number of process neighbors *(new to this work)*



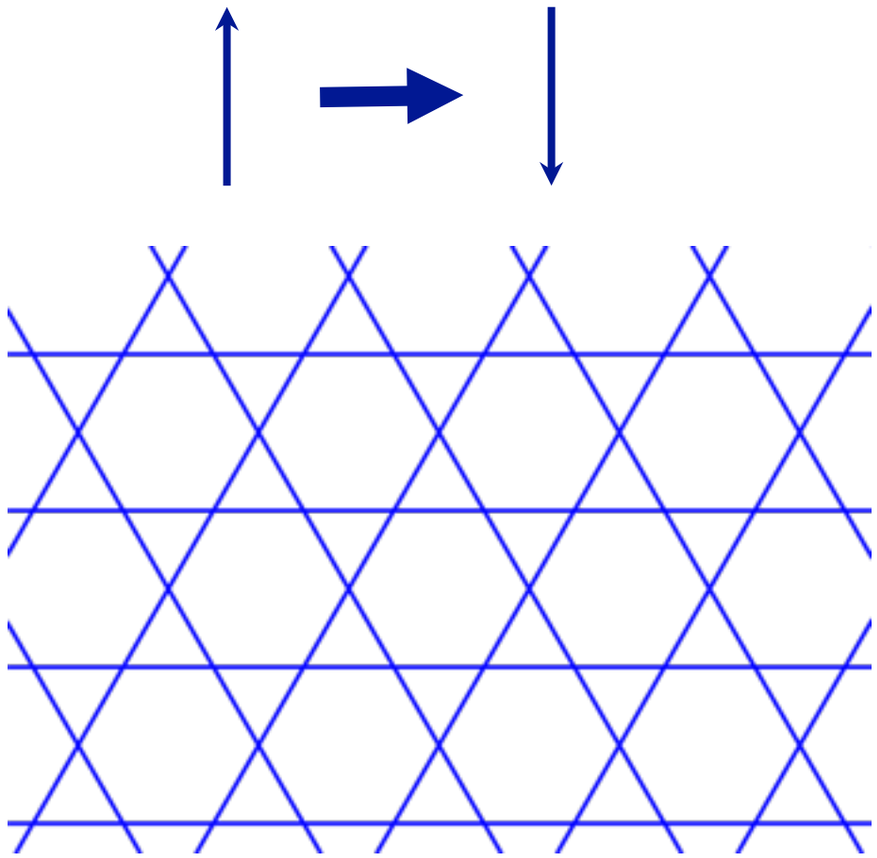
Algorithm 1 *Lanczos iteration*

1: $v_1 \leftarrow$ random vector with norm 1
2: $v_0 \leftarrow 0$
3: $\beta_1 \leftarrow 0$
4: **for** $j = 1, \ldots, r$ **do**
5: $\quad w_j \leftarrow Hv_j - \beta_j v_{j-1}$
6: $\quad \alpha_j \leftarrow (w_j, v_j)$
7: $\quad \beta_{j+1} \leftarrow \|w_j\|$
8: $\quad v_{j+1} \leftarrow w_j / \beta_{j+1}$
9: **end for**

Fundamental ED problem

$$H = J \sum S_i^z S_j^z + \Gamma \sum S_i^x$$

- Any lattice with n sites, 2^n states
- Lanczos eigensolver
- Large, sparse symmetric mat-vec
- Operator has integer operations
- Very irregular sparsity, but
- Limited number of process neighbors *(new to this work)*
- Symmetries considered in some models: smaller complexity at cost of more communication



Algorithm 1 *Lanczos iteration*

1: $v_1 \leftarrow$ random vector with norm 1
2: $v_0 \leftarrow 0$
3: $\beta_1 \leftarrow 0$
4: **for** $j = 1, \ldots, r$ **do**
5: $\quad w_j \leftarrow Hv_j - \beta_j v_{j-1}$
6: $\quad \alpha_j \leftarrow (w_j, v_j)$
7: $\quad \beta_{j+1} \leftarrow \|w_j\|$
8: $\quad v_{j+1} \leftarrow w_j / \beta_{j+1}$
9: **end for**

Benchmark code: simplest “SPIN” model

Benchmark code: simplest “SPIN” model

Loop for MAX_ITER
- **Reduction B** (MPI_Reduce)
- **L3** (local work, normalize v1)
- **Loop over rounds of msgs**
  - **MSG_NB** (MPI_Isend,upc_memput(_nbi))
  - **L4** (work on local matrix, only 1st iteration)
  - **SYNC** (no-op, upc_fence)
  - **L7** (manage msg reception and do remote work)
- **L8** (local work, A norm)
- **Reduction A** (MPI_Reduce)
- **L9** (local work, B norm)

Benchmark code: simplest "SPIN" model

Loop for MAX_ITER
- **Reduction B** (MPI_Reduce)
- **L3** (local work, normalize v1)
- **Loop over rounds of msgs**
  - **MSG_NB** (MPI_Isend,upc_memput(_nbi))
  - **L4** (work on local matrix, only 1st iteration)
  - **SYNC** (no-op, upc_fence)
  - **L7** (manage msg reception and do remote work)
- **L8** (local work, A norm)
- **Reduction A** (MPI_Reduce)
- **L9** (local work, B norm)

Loops:

- L3: Initialize array
- L4: Local mat-vec
- L6/7: Off process mat-vec
- L8: Alpha calculation
- L9: Beta calculation

Benchmark code: simplest "SPIN" model

Loop for MAX_ITER
- **Reduction B** (MPI_Reduce)
- **L3** (local work, normalize v1)
- **Loop over rounds of msgs**
  - **MSG_NB** (MPI_Isend,upc_memput(_nbi))
  - **L4** (work on local matrix, only 1st iteration)
  - **SYNC** (no-op, upc_fence)
  - **L7** (manage msg reception and do remote work)
- **L8** (local work, A norm)
- **Reduction A** (MPI_Reduce)
- **L9** (local work, B norm)

Loops:
- **L3: Initialize array**
- **L4: Local mat-vec**
- **L6/7: Off process mat-vec**
- **L8: Alpha calculation**
- **L9: Beta calculation**

Code structure

```
void execute(ed){
  L1: for (i=0; i<2^(n-m); i++)
        b += v1[i]*v1[i]
  L2: for(iter=0; iter<max_iters; iter++)
        MPI_Barrier
        MPI_Allreduce of b
      L3: for (i=0; i<2^(n-m); i++)                    L3
            v1[i] *= f1(iter,b)
      MPI_Isend/MPI_Irecv to vv1 using handler1
      // local contribution, no comm
      L4: for (i=0; i<2^(n-m); i++)                    L4
            v2[i] = a(i)*v1[i]
          L5: for (k=0; k<n-m; k++)
                v2[i] += g*v1[f2(i,k)]
      // remote contribution, comm
      L6: for (k=n-m; k<n; k++) // m iterations         L6
        MPI_Isend/MPI_Irecv to vv2 using handler2
        MPI_Wait on handler1
        L7: for (i=0; i<2^(n-m); i++)
              v2[i] += g*vv1[i]
        swap(vv1,vv2) // Pointer swap, no memcpy
        handler1 = handler2
      L8: for(i=0; i<2^(n-m); i++)                     L8
            a += v1[i]*v2[i]
        MPI_Barrier
        MPI_Allreduce of a
      L9: for(i=0; i<2^(n-m); i++)                     L9
            v2[i] = f3(i,iter,a)
            b += v2[i]*v2[i]
        swap (v0,v1,v2)
}
```

execute

SPIN single core/socket/node comparisons

- Loop-based OMP directives: performance worse than MPI-only
- Task-based OpenMP/MPI implementation by Fourestey/Stringfellow

SPIN single core/socket/node comparisons

- Loop-based OMP directives: performance worse than MPI-only
- Task-based OpenMP/MPI implementation by Fourestey/Stringfellow

System name	Rivera	Castor	Sandy
Processor	AMD 6274	Intel E5-2680	Intel X5650
Nickname	Interlagos	Westmere	Sandybridge
Cores/Socket	16	6	8
Sockets/Node	2	2	2
Hyperthreading	no	unenabled	yes (2)
Compiler	Open64	Intel	Intel
Core time (s.)	754 (1T)	280 (1T)	227 (1T)
Socket time (s.)	74 (15T)	51 (6T)	29 (16T)
Node time (s.)	38 (31T)	26 (12T)	15 (32T)

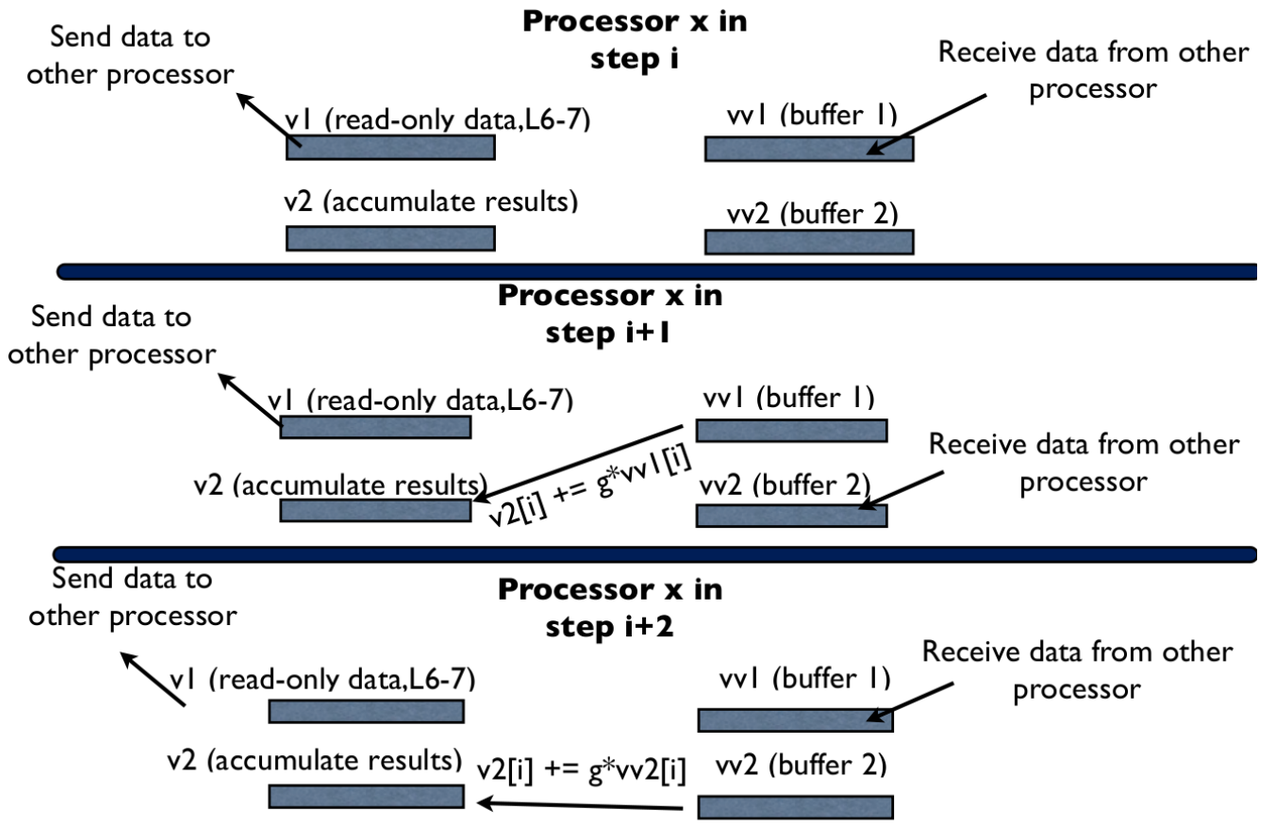
Multi-buffering concept

Multi-buffering concept

Double buffering

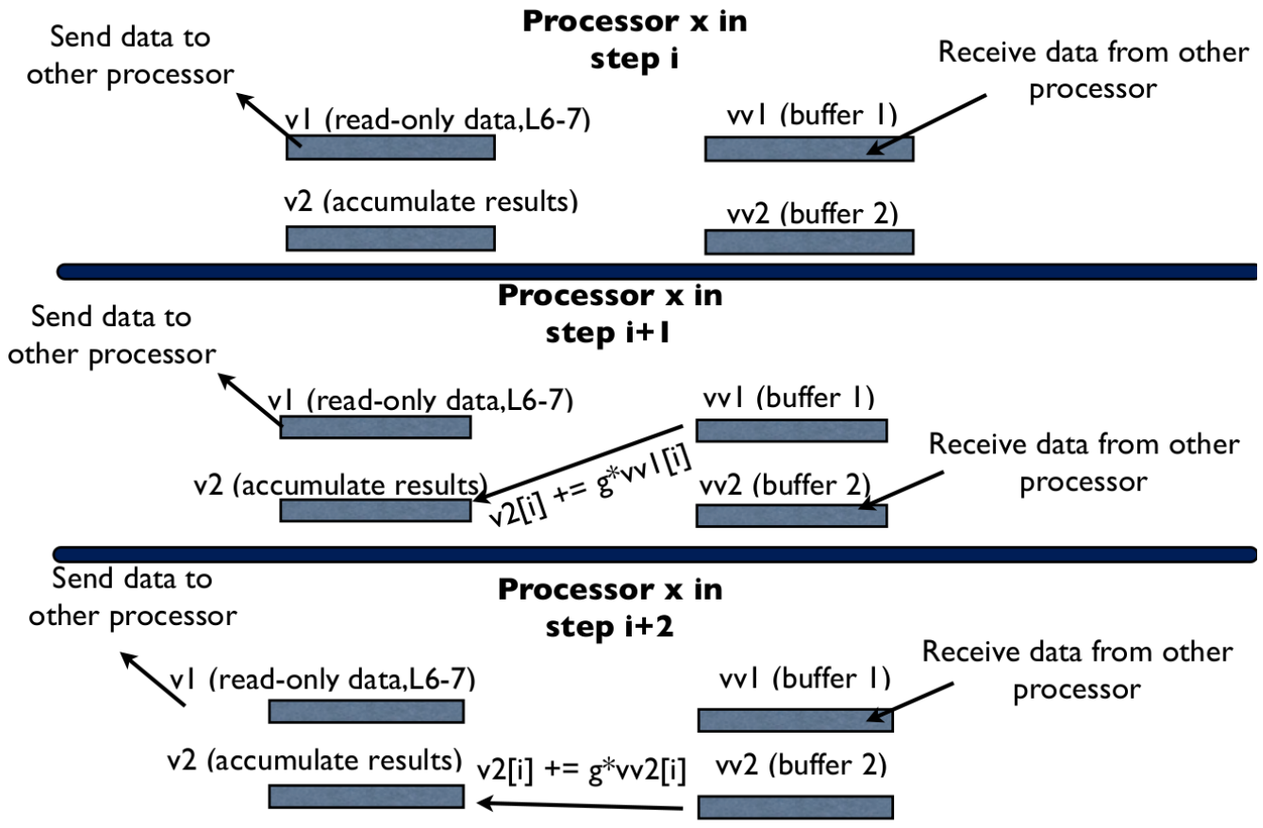
Multi-buffering concept

Double buffering



Multi-buffering concept

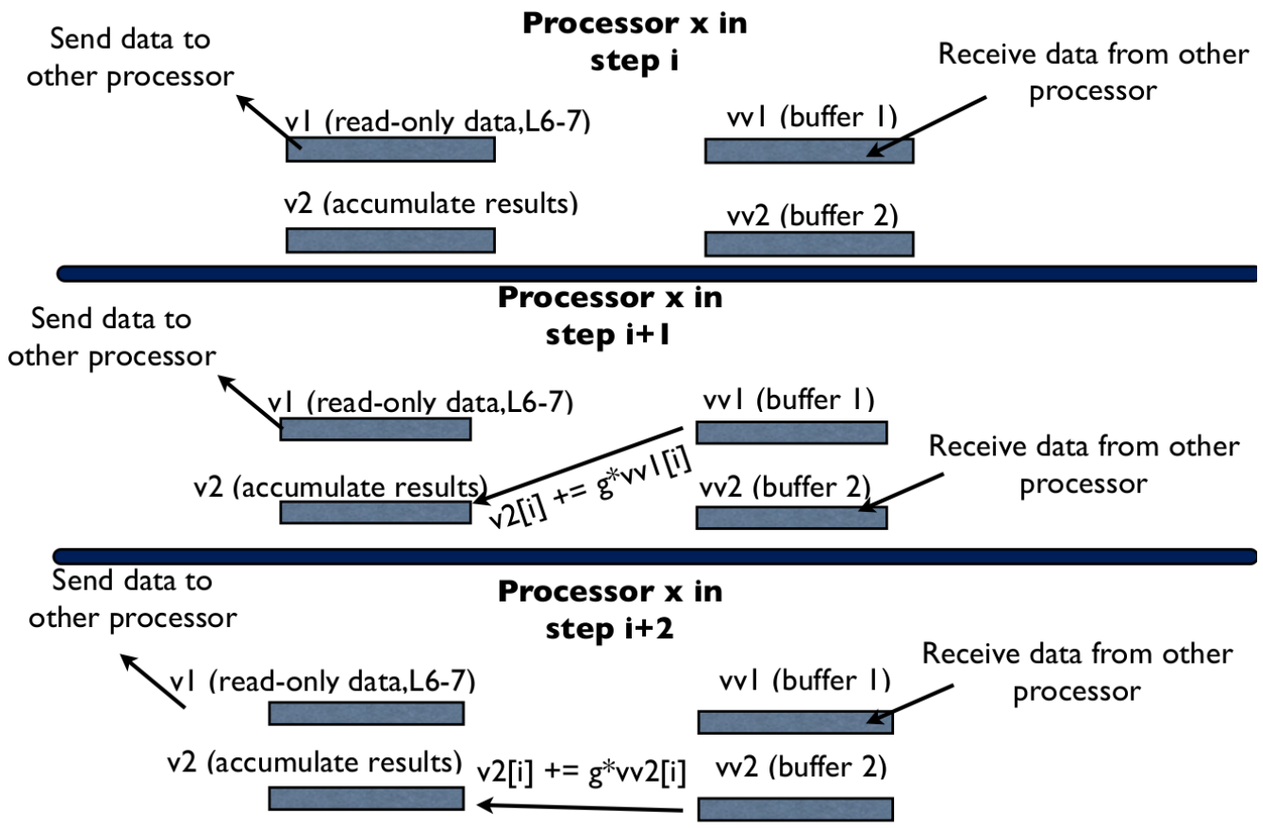
Double buffering



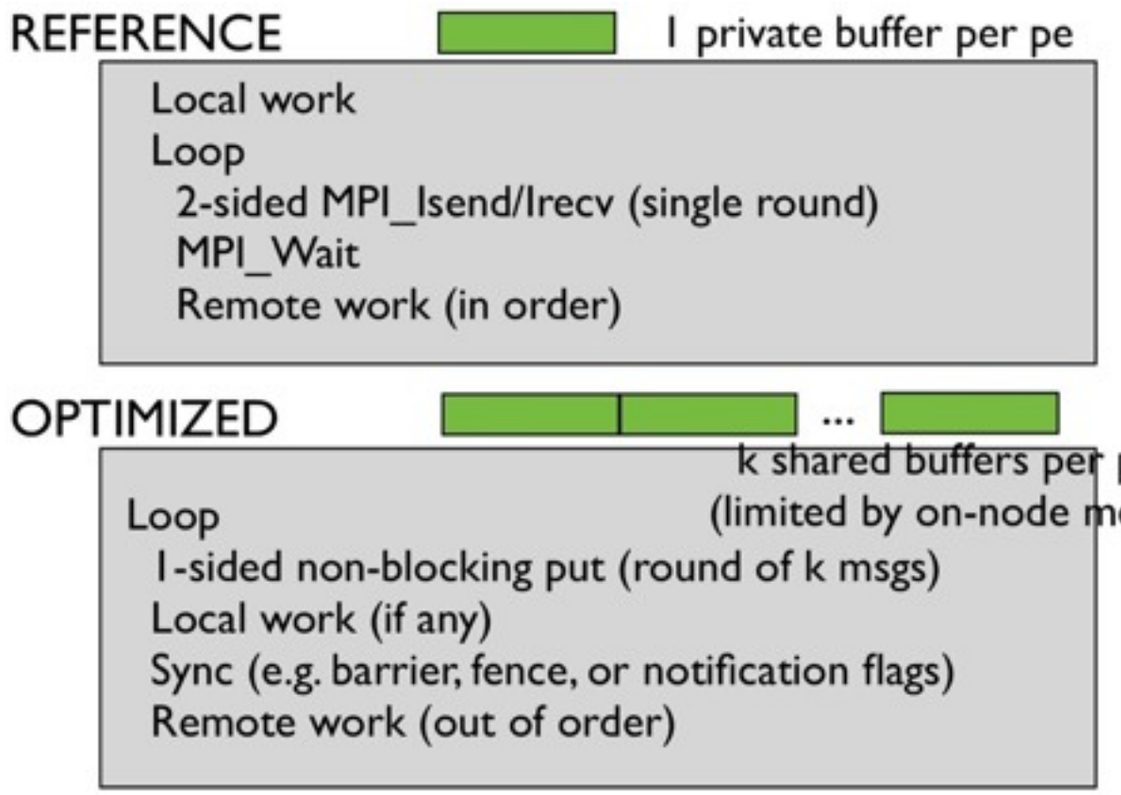
Multi-buffering

Multi-buffering concept

Double buffering



Multi-buffering



UPC “Elegant” Implementation

UPC "Elegant" Implementation

```
struct ed_s { ...
        shared double *v0, *v1, *v2;        /* vectors */
        shared double *swap;                /* for swapping vectors */
};
                :
for (iter = 0; iter < ed->max_iter; ++iter) {
                  :
                upc_barrier(0);
                /* matrix vector multiplication */
                upc_forall (s = 0; s < ed->nlstates; ++s; &(ed->v1[s]) ) {
                        /* diagonal part */
                        ed->v2[s] = diag(s, ed->n, ed->j) * ed->v1[s];
                        /* offdiagonal part */
                        for (k = 0; k < ed->n; ++k) {
                                s1 = flip_state(s, k);
                                ed->v2[s] += ed->gamma * ed->v1[s1];
                        }
                }
                 :
                /* Calculate alpha */
                /* Calculate beta */
        }
}
```

Inelegant **UPC versions**

Inelegant UPC versions

Inelegant 1

```
shared[NBLOCK] double vtmp[THREADS*NBLOCK];
  :
for (i = 0; i < NBLOCK; ++i) vtmp[i+MYTHREAD*NBLOCK] = ed->v1[i];
upc_barrier(1);
for (i = 0; i < NBLOCK; ++i) ed->vv1[i] = vtmp[i+(ed->from_nbs[0]*NBLOCK)];
  :
upc_barrier(2);
```

Inelegant 2

```
shared[NBLOCK] double vtmp[THREADS*NBLOCK];
  :
upc_memput( &vtmp[MYTHREAD*NBLOCK], ed->v1, NBLOCK*sizeof(double) );
upc_barrier(1);
upc_memget( ed->vv1, &vtmp[ed->from_nbs[0]*NBLOCK], NBLOCK*sizeof(double) );
  :
upc_barrier(2);
```

UPC *Inelegant3:* use double buffers and upc_put

UPC *Inelegant3:* use double buffers and upc_put

```
shared[NBLOCK] double vtmp1[THREADS*NBLOCK];
shared[NBLOCK] double vtmp2[THREADS*NBLOCK];
:
upc_memput( &vtmp1[ed->to_nbs[0]*NBLOCK], ed->v1, NBLOCK*sizeof(double) );
upc_barrier(1);
:
 if ( mode == 0 ) {
   upc_memput( &vtmp2[ed->to_nbs[neighb]*NBLOCK], ed->v1, NBLOCK*sizeof(double) );
 } else {
   upc_memput( &vtmp1[ed->to_nbs[neighb]*NBLOCK], ed->v1, NBLOCK*sizeof(double) );
 }
  :
if ( mode == 0 ) {
  for (i = 0; i < ed->nlstates; ++i) { ed->v2[i] += ed->gamma * vtmp1[i+MYTHREAD*NBLOCK]; }
  mode = 1;
} else {
  for (i = 0; i < ed->nlstates; ++i) { ed->v2[i] += ed->gamma * vtmp2[i+MYTHREAD*NBLOCK]; }
  mode = 0;
}
upc_barrier(2);
```

Other message passing paradigms

MPI-2: One-sided PUT

```
MPI_Put(ed->v1, ed->nlstates, MPI_DOUBLE, ed->to_nbs[0], 0, ed->nlstates, MPI_DOUBLE, win1);
MPI_Win_fence( 0, win1);
```

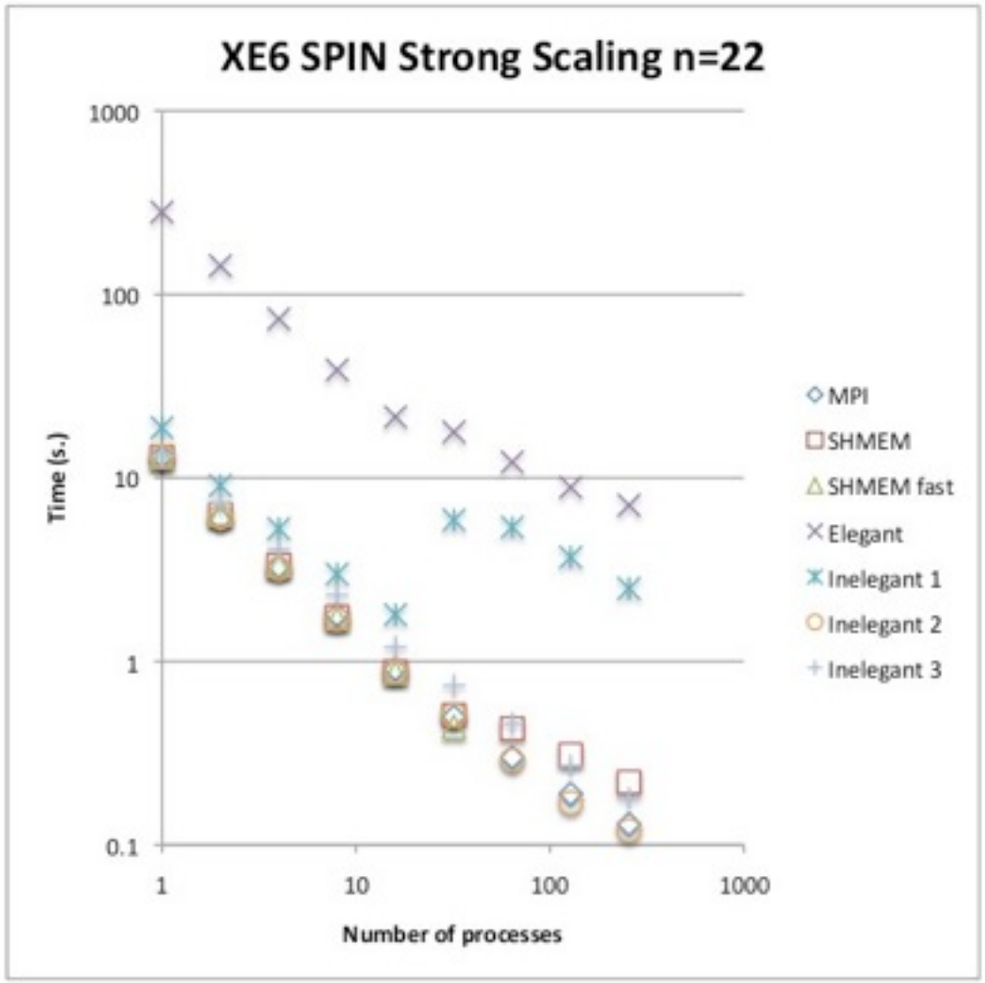
SHMEM: non-blocking PUT

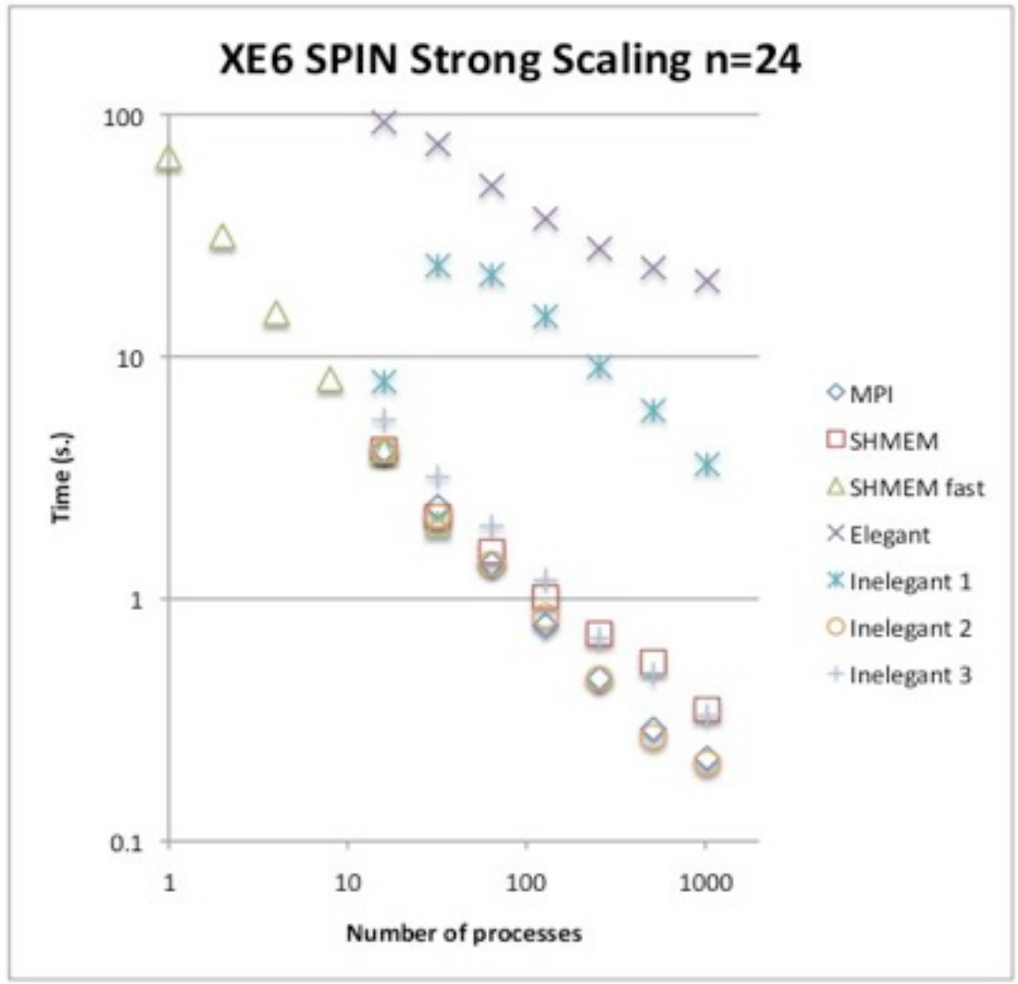
```
vtmp1 = (double *) shmalloc(ed->nlstates*sizeof(double));
  :
shmem_barrier_all();
shmem_double_put_nb(vtmp1, ed->v1, ed->nlstates, ed->from_nbs[neighb], NULL);
```

SHMEM "fast": non-blocking PUT, local wait only

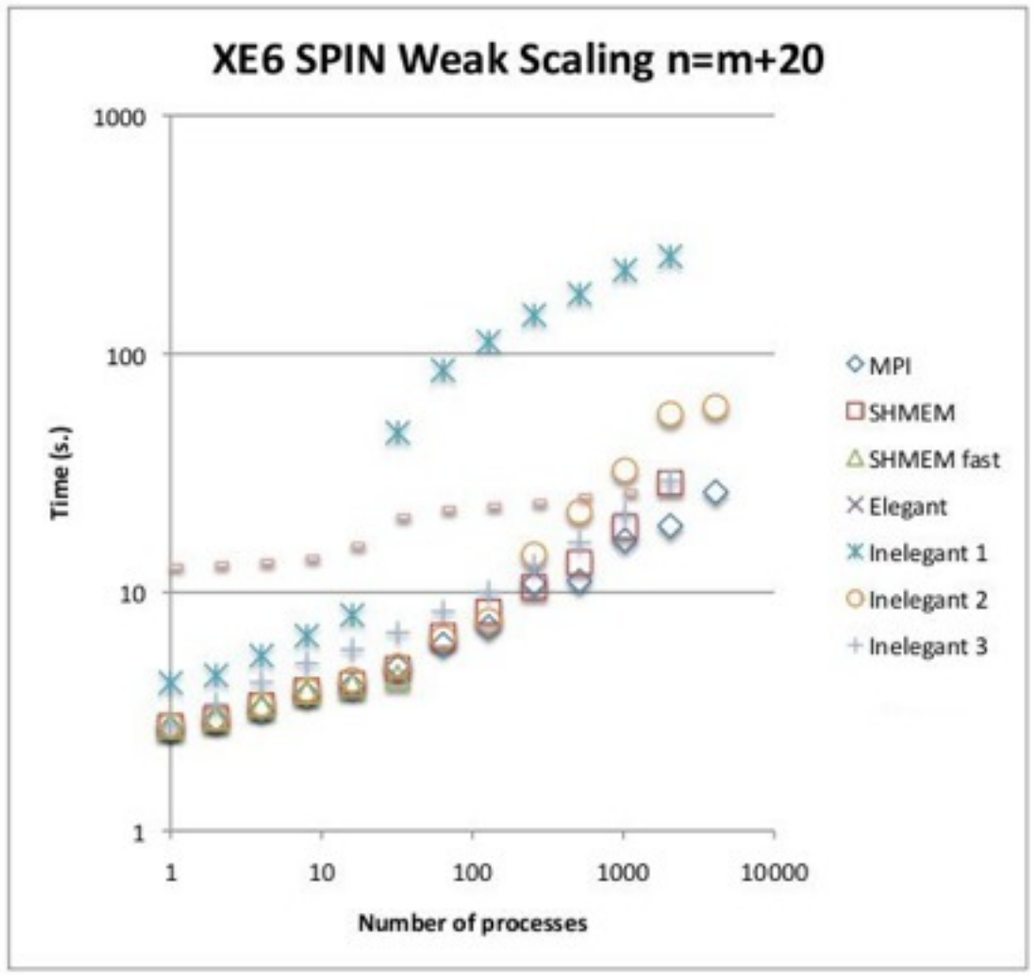
```
ed->v1[ed->nlstates] = ((double) ed->rank); /* sentinel */
for (l = 0; l < ed->m; ++l) {
  offset = l*(ed->nlstates+1); /* Offset into buffer */
  shmem_double_put_nb(&vtmp[offset],ed->v1, ed->nlstates+1,ed->to_nbs[l],NULL);
}
            :
tag = vtmp[offset+ed->nlstates];
while (tag != (double) ed->from_nbs[k-ed->nm]) { /* spin */
  tag = vtmp[offset+ed->nlstates];
}
for (i = offset, j=0; i < offset+ed->nlstates; ++i, ++j) {
   ed->v2[j] += ed->gamma * vtmp[i];
}
vtmp[l*(ed->nlstates+1)+ed->nlstates]=((double)-1); /*reset*/
```

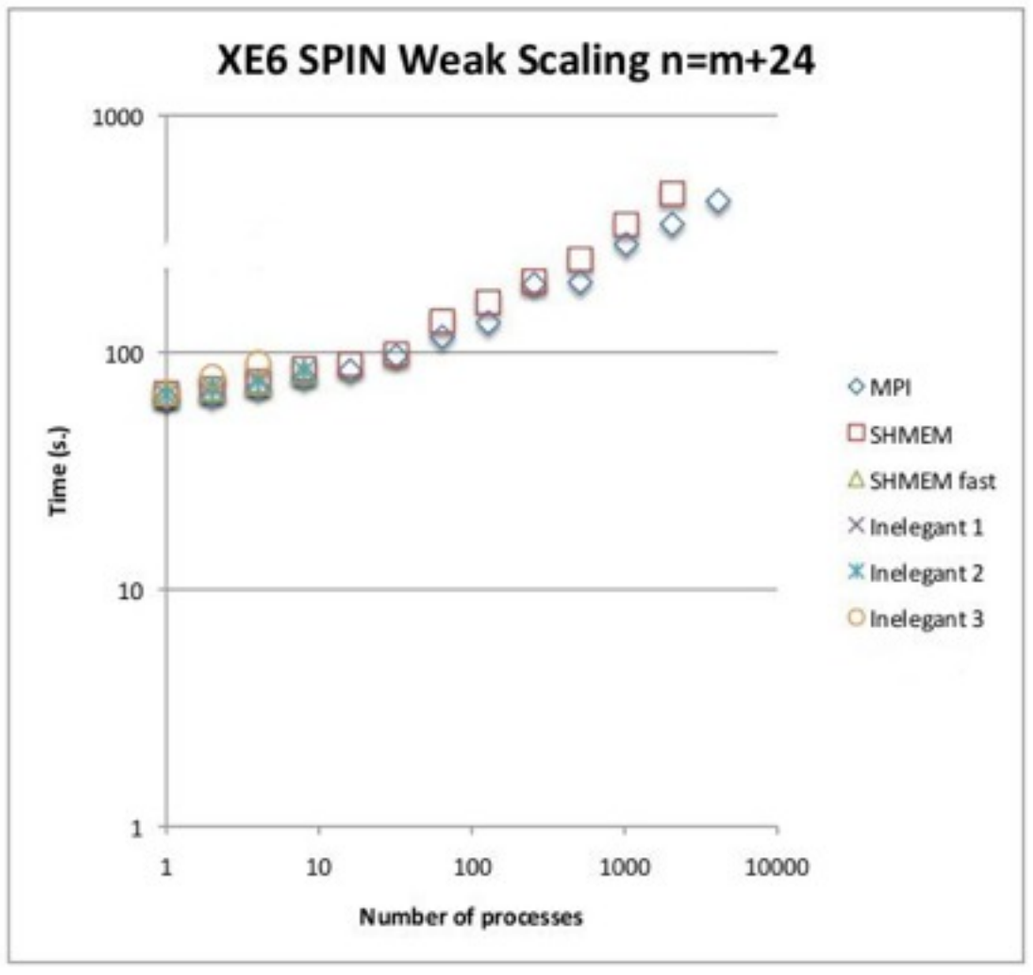
SPIN strong scaling: Cray XE6, n=22,24; 10 iter.





SPIN weak scaling: Cray XE6/Gemini, 10 iterations





Can UPC perform better than MPI two-sided?

Can UPC perform better than MPI two-sided?

- **Work:** original MPI two-sided version with double buffering

Can UPC perform better than MPI two-sided?

- **Work:** original MPI two-sided version with double buffering
- **Ref_MPI:** naive single buffered version

Can UPC perform better than MPI two-sided?

- **Work:** original MPI two-sided version with double buffering
- **Ref_MPI:** naive single buffered version
- **Opt_MPI:** multiple round-robin buffers utilizing MPI_Isend/Irecv

Can UPC perform better than MPI two-sided?

- **Work:** original MPI two-sided version with double buffering
- **Ref_MPI:** naive single buffered version
- **Opt_MPI:** multiple round-robin buffers utilizing MPI_Isend/Irecv
- **Opt_UPC_Fence:** blocking upc_memput with single fence

Can UPC perform better than MPI two-sided?

- **Work:** original MPI two-sided version with double buffering
- **Ref_MPI:** naive single buffered version
- **Opt_MPI:** multiple round-robin buffers utilizing MPI_Isend/Irecv
- **Opt_UPC_Fence:** blocking upc_memput with single fence
- **Opt_UPC_Fence_each:** blocking upc_memput with fence for each message

Can UPC perform better than MPI two-sided?

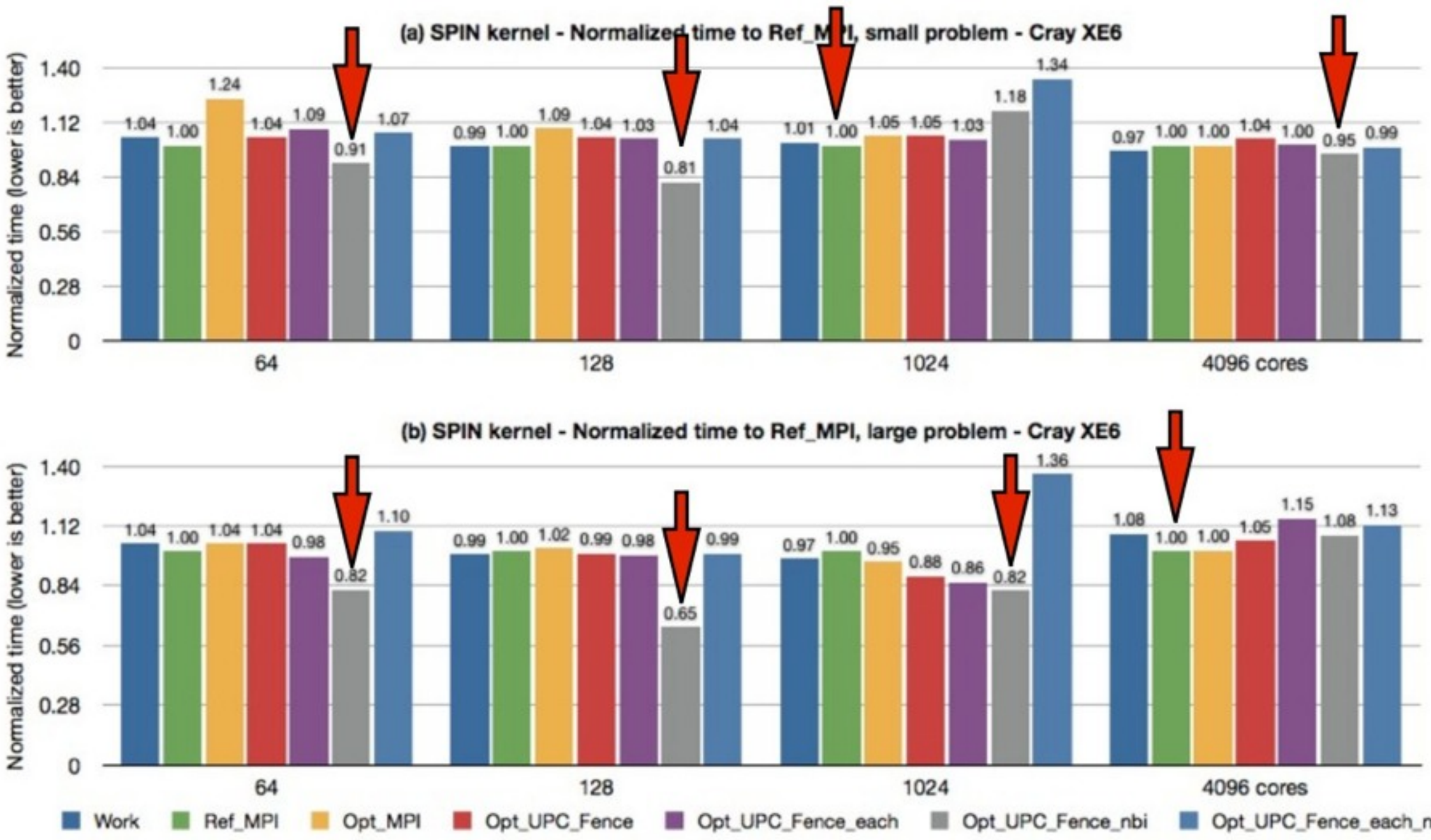
- **Work:** original MPI two-sided version with double buffering
- **Ref_MPI:** naive single buffered version
- **Opt_MPI:** multiple round-robin buffers utilizing MPI_Isend/Irecv
- **Opt_UPC_Fence:** blocking upc_memput with single fence
- **Opt_UPC_Fence_each:** blocking upc_memput with fence for each message
- **Opt_UPC_Fence_nbi:** Cray-specific implicit non-blocking memput with a single fence

Can UPC perform better than MPI two-sided?

- **Work:** original MPI two-sided version with double buffering
- **Ref_MPI:** naive single buffered version
- **Opt_MPI:** multiple round-robin buffers utilizing MPI_Isend/Irecv
- **Opt_UPC_Fence:** blocking upc_memput with single fence
- **Opt_UPC_Fence_each:** blocking upc_memput with fence for each message
- **Opt_UPC_Fence_nbi:** Cray-specific implicit non-blocking memput with a single fence
- **Opt_UPC_Fence_each_nbi:** Cray-specific implicit non-blocking memput with fence for each message

Optimized SPIN normed performance: Cray XE6

Optimized SPIN normed performance: Cray XE6



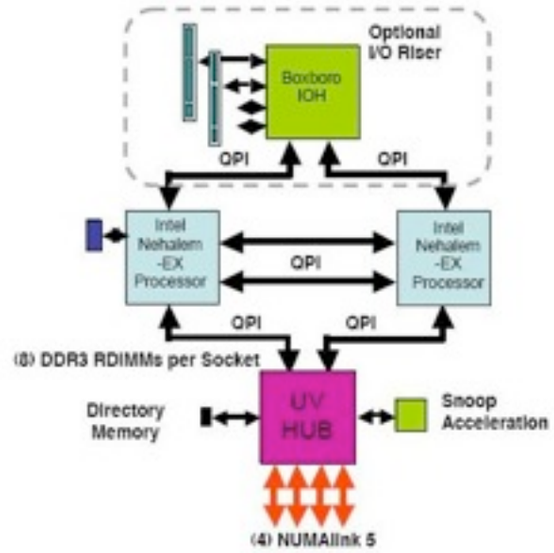
Optimized SPIN normed performance: SGI UV1000

Optimized SPIN normed performance: SGI UV1000

- Shared-memory, extensible to 256 sockets
- Fat node NUMAlink interconnect

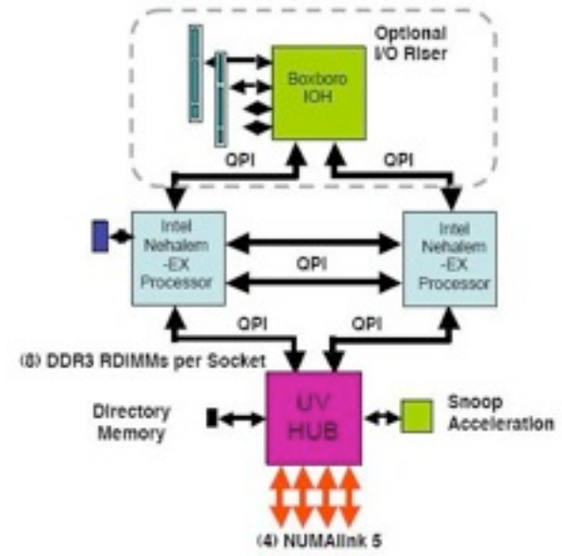
Optimized SPIN normed performance: SGI UV1000

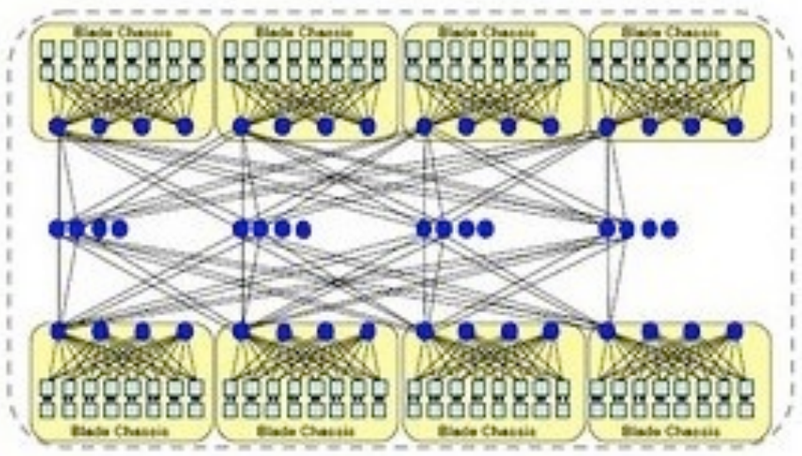
- Shared-memory, extensible to 256 sockets
- Fat node NUMAlink interconnect



Optimized SPIN normed performance: SGI UV1000

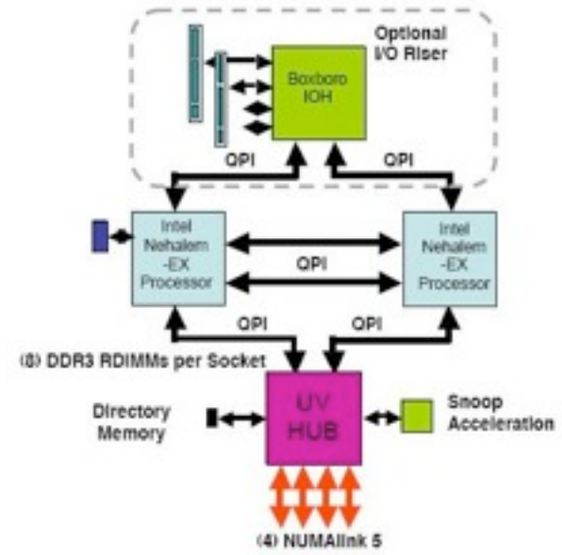
- Shared-memory, extensible to 256 sockets
- Fat node NUMAlink interconnect

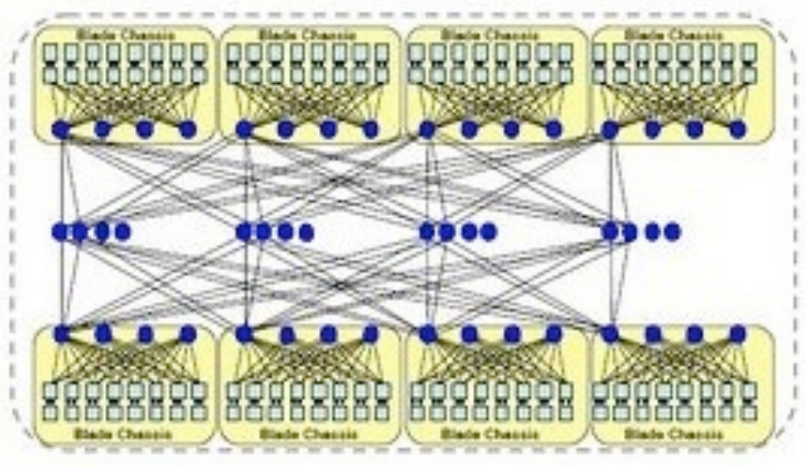


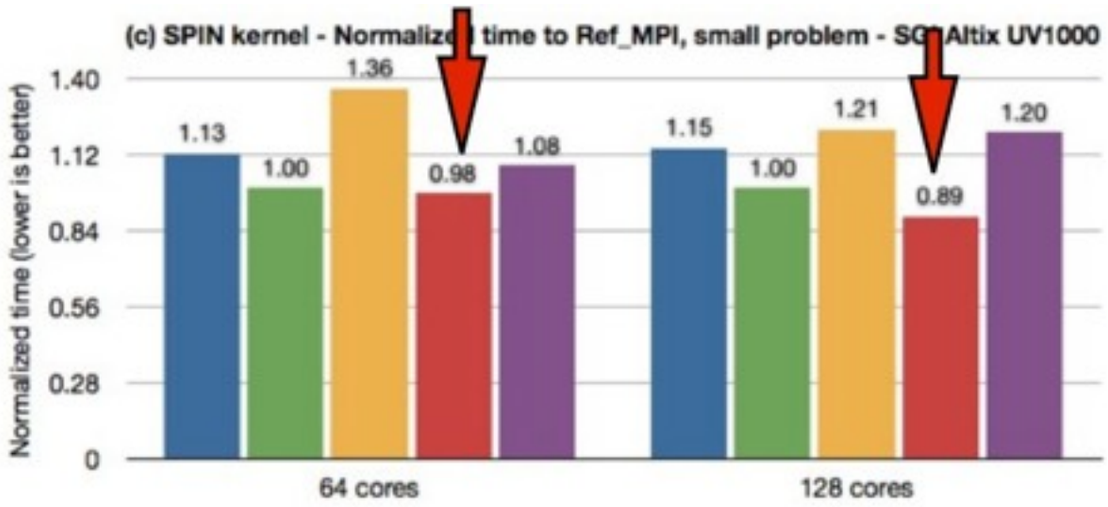


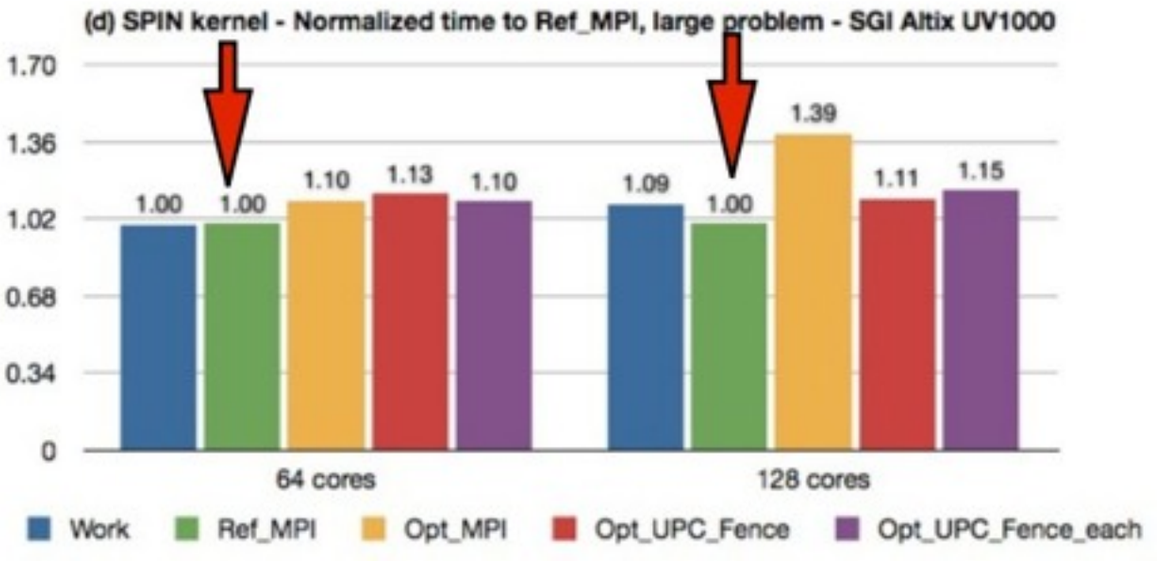
Optimized SPIN normed performance: SGI UV1000

- Shared-memory, extensible to 256 sockets
- Fat node NUMAlink interconnect



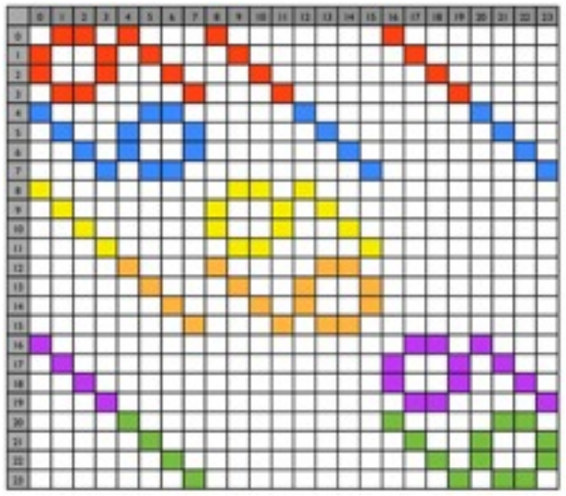




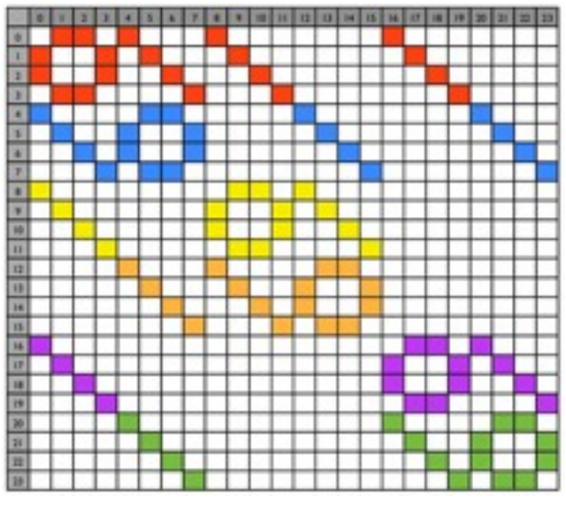


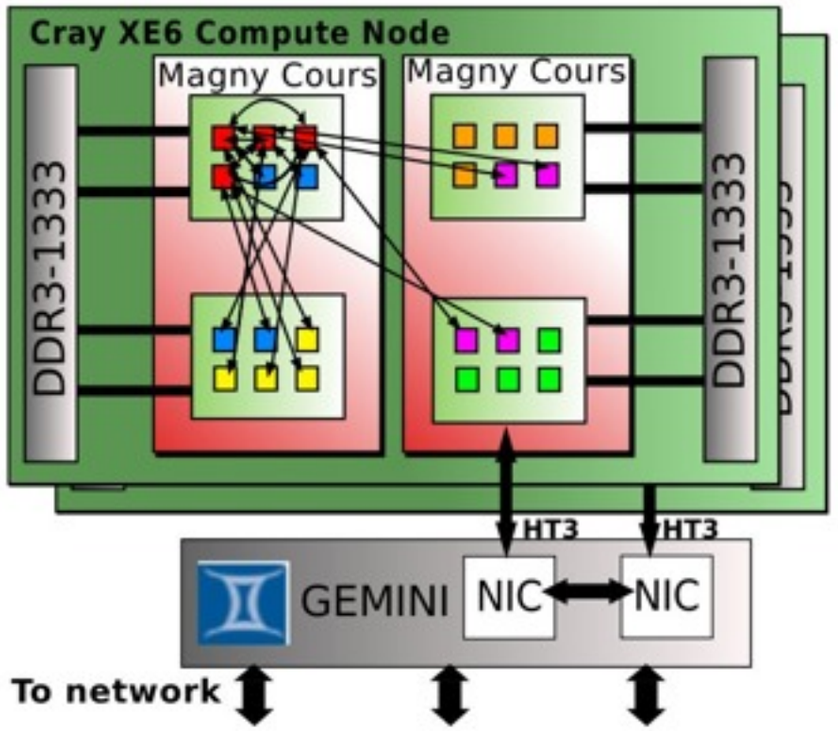
Optimizing task placement (XE6, Magny Cours)

Optimizing task placement (XE6, Magny Cours)

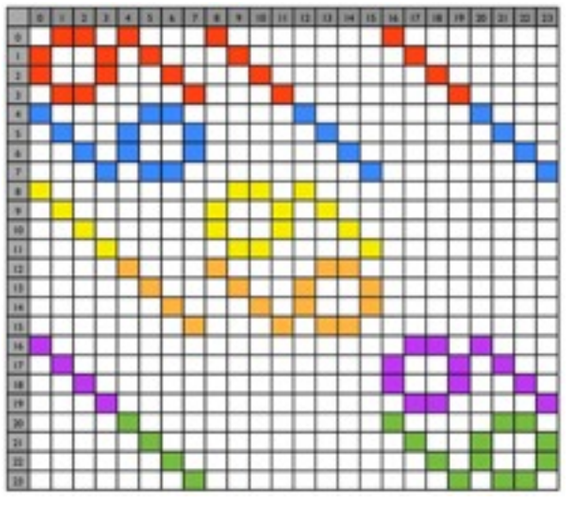


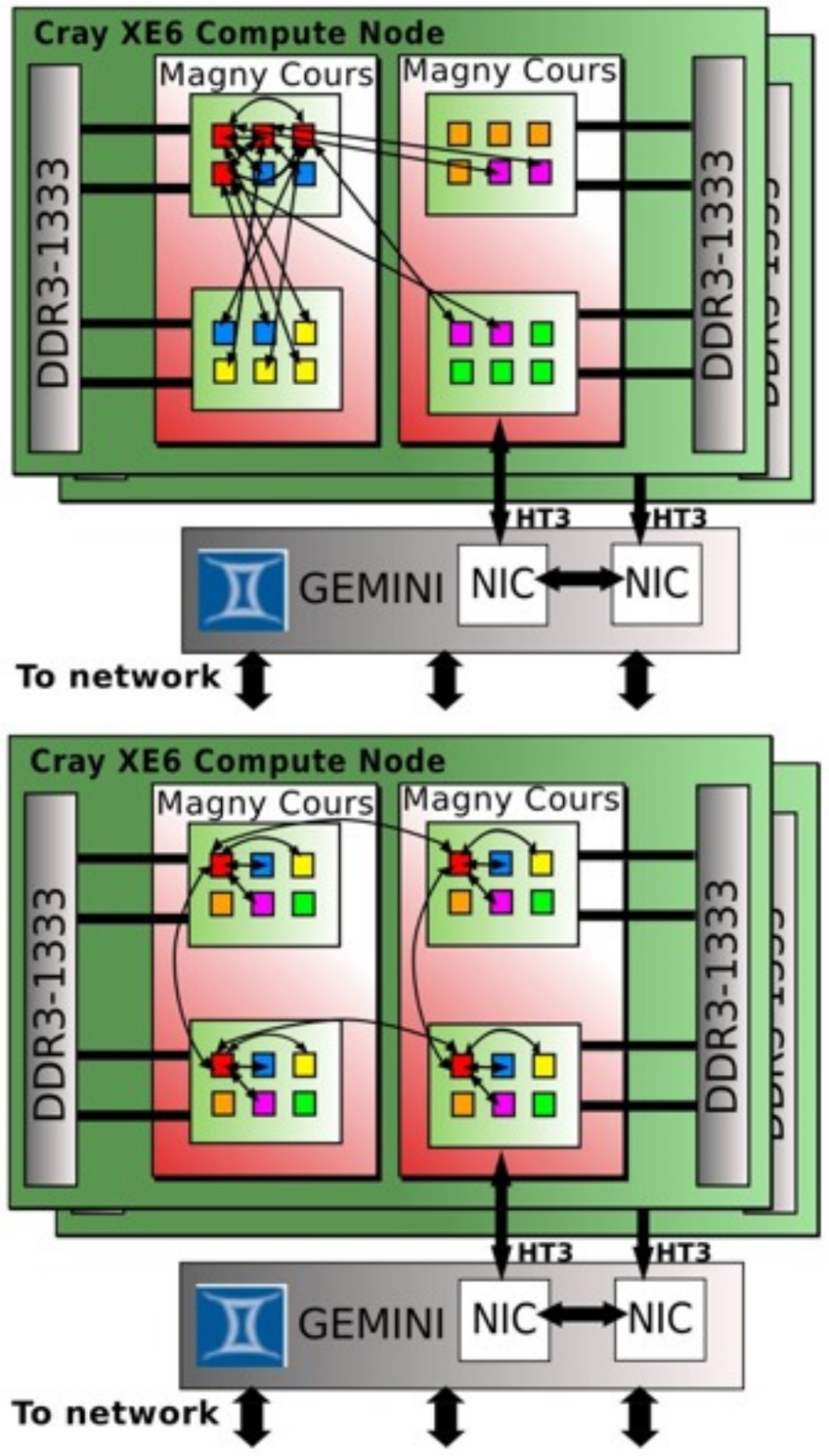
Optimizing task placement (XE6, Magny Cours)



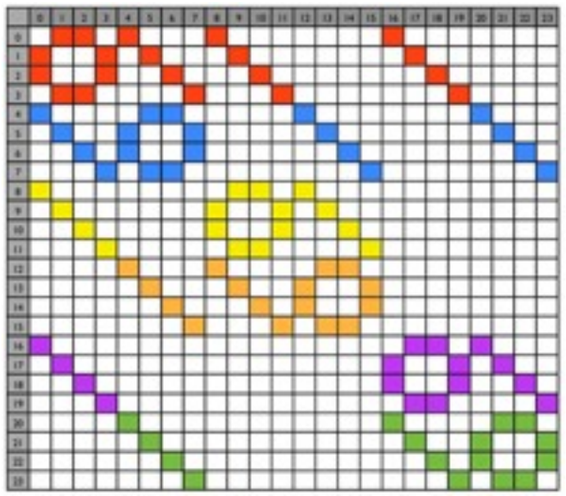


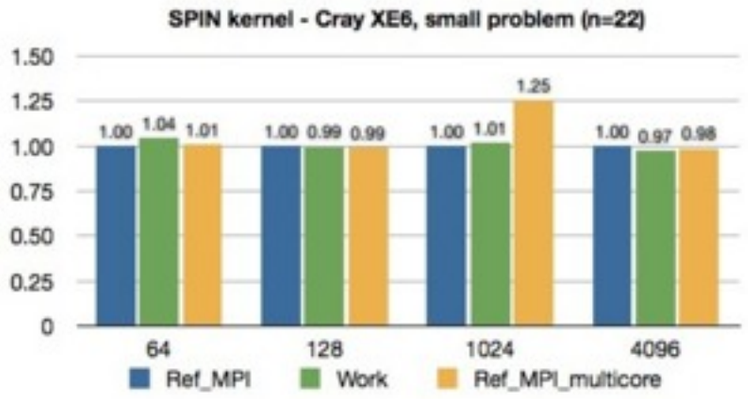
Optimizing task placement (XE6, Magny Cours)

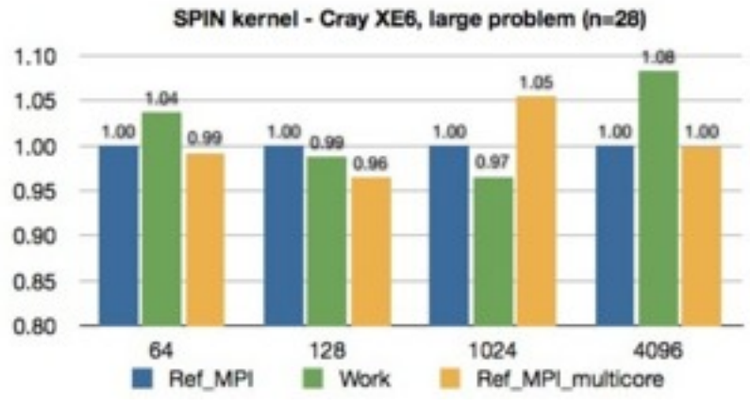


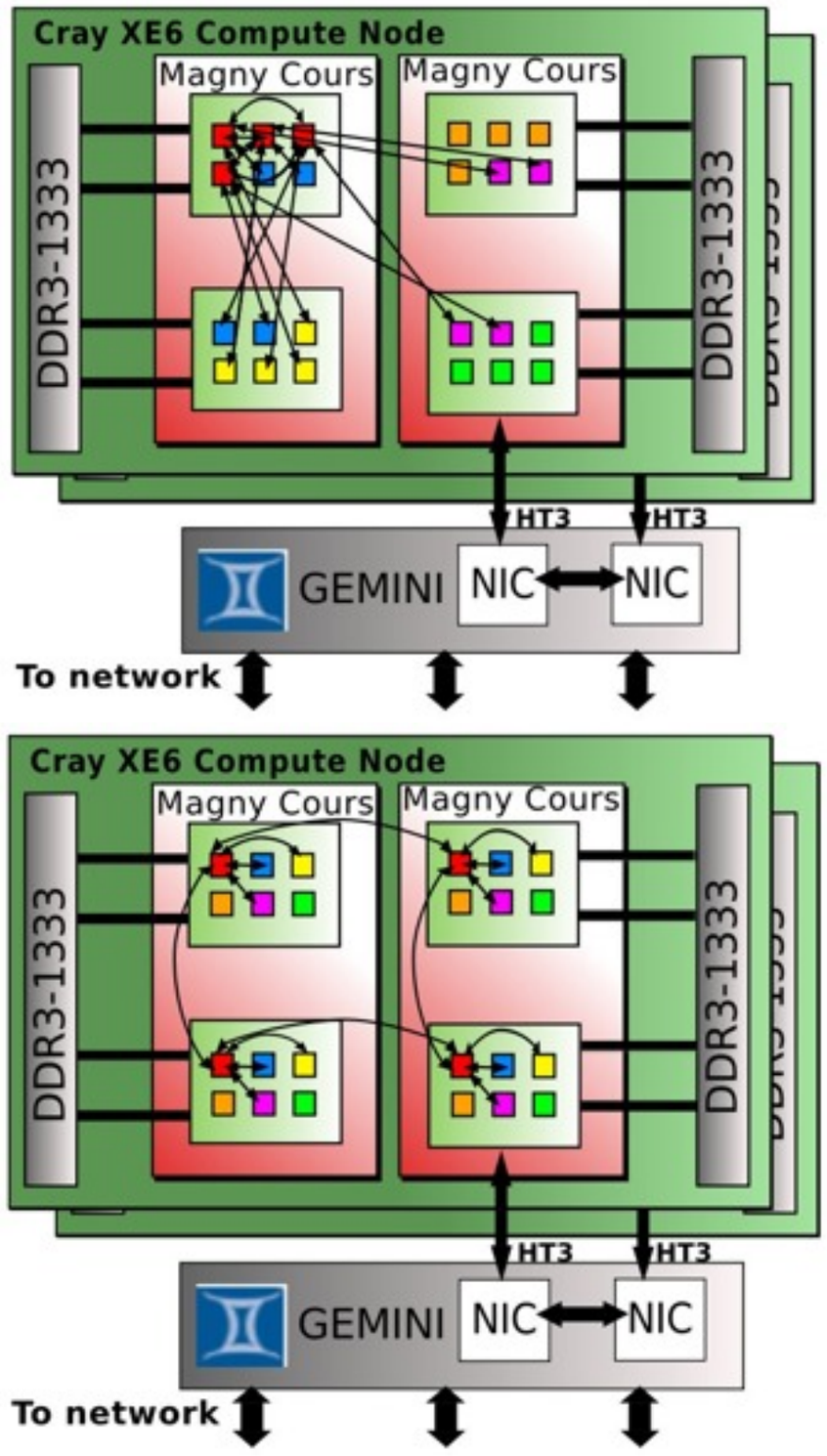


Optimizing task placement (XE6, Magny Cours)









SPIN experiences

SPIN experiences

- SPIN is not computationally intensive

SPIN experiences

- SPIN is not computationally intensive
- Contain integer operations, e.g., bit shifting

SPIN experiences

- SPIN is not computationally intensive
- Contain integer operations, e.g., bit shifting
- Fundamentally a benchmark of the interconnection network

SPIN experiences

- SPIN is not computationally intensive
- Contain integer operations, e.g., bit shifting
- Fundamentally a benchmark of the interconnection network
- MPI two-sided performing better than (nearly) all other communication paradigms, including SHMEM, MPI-2

SPIN experiences

- SPIN is not computationally intensive
- Contain integer operations, e.g., bit shifting
- Fundamentally a benchmark of the interconnection network
- MPI two-sided performing better than (nearly) all other communication paradigms, including SHMEM, MPI-2
- Work by A. Tineo showed that UPC+optimizations can attain better performance in some cases

SPIN experiences

- SPIN is not computationally intensive
- Contain integer operations, e.g., bit shifting
- Fundamentally a benchmark of the interconnection network
- MPI two-sided performing better than (nearly) all other communication paradigms, including SHMEM, MPI-2
- Work by A. Tineo showed that UPC+optimizations can attain better performance in some cases
- Simplistic OpenMP/MPI hybrid performed not better than MPI

SPIN experiences

- SPIN is not computationally intensive
- Contain integer operations, e.g., bit shifting
- Fundamentally a benchmark of the interconnection network
- MPI two-sided performing better than (nearly) all other communication paradigms, including SHMEM, MPI-2
- Work by A. Tineo showed that UPC+optimizations can attain better performance in some cases
- Simplistic OpenMP/MPI hybrid performed not better than MPI
- Task-based OpenMP/MPI implementation by Fourestey/ Stringfellow did show slightly better performance (n=28 test case)

SPIN experiences

- SPIN is not computationally intensive
- Contain integer operations, e.g., bit shifting
- Fundamentally a benchmark of the interconnection network
- MPI two-sided performing better than (nearly) all other communication paradigms, including SHMEM, MPI-2
- Work by A. Tineo showed that UPC+optimizations can attain better performance in some cases
- Simplistic OpenMP/MPI hybrid performed not better than MPI
- Task-based OpenMP/MPI implementation by Fourestey/ Stringfellow did show slightly better performance (n=28 test case)

MPI Processes	MPI-only (s.)	2 Threads (s.)	4 Threads (s.)
4096	17.4		
2048	28.1	16.6	
1024	48.9	25.1	14.4

Exact Diagonalization: HP2C Implementation

Exact Diagonalization: HP2C Implementation

Full ED application implemented within the High Performance and High Productivity Computing Initiative (`www.hp2c.ch`)

Exact Diagonalization: HP2C Implementation

Full ED application implemented within the High Performance and High Productivity Computing Initiative (`www.hp2c.ch`)

- Symmetries (reduces complexity at cost of more communication)

Exact Diagonalization: HP2C Implementation

Full ED application implemented within the High Performance and High Productivity Computing Initiative (`www.hp2c.ch`)

- Symmetries (reduces complexity at cost of more communication)
- Supports multiple one- and two-dimensional lattices

Exact Diagonalization: HP2C Implementation

Full ED application implemented within the High Performance and High Productivity Computing Initiative (`www.hp2c.ch`)

- Symmetries (reduces complexity at cost of more communication)
- Supports multiple one- and two-dimensional lattices
- Multiple quantum models

Exact Diagonalization: HP2C Implementation

Full ED application implemented within the High Performance and High Productivity Computing Initiative (`www.hp2c.ch`)

- Symmetries (reduces complexity at cost of more communication)
- Supports multiple one- and two-dimensional lattices
- Multiple quantum models
  - Heisenberg

Exact Diagonalization: HP2C Implementation

Full ED application implemented within the High Performance and High Productivity Computing Initiative (`www.hp2c.ch`)

- Symmetries (reduces complexity at cost of more communication)
- Supports multiple one- and two-dimensional lattices
- Multiple quantum models
  - Heisenberg
  - Fendley (computationally intensive, unlike SPIN benchmark)

Exact Diagonalization: HP2C Implementation

Full ED application implemented within the High Performance and High Productivity Computing Initiative (`www.hp2c.ch`)

- Symmetries (reduces complexity at cost of more communication)
- Supports multiple one- and two-dimensional lattices
- Multiple quantum models
  - Heisenberg
  - Fendley (computationally intensive, unlike SPIN benchmark)
  - FQHE (computationally intensive)

Exact Diagonalization: HP2C Implementation

Full ED application implemented within the High Performance and High Productivity Computing Initiative (`www.hp2c.ch`)

- Symmetries (reduces complexity at cost of more communication)
- Supports multiple one- and two-dimensional lattices
- Multiple quantum models
  - Heisenberg
  - Fendley (computationally intensive, unlike SPIN benchmark)
  - FQHE (computationally intensive)

- OMP/MPI Implementation
  (simple loop-based directives)

Exact Diagonalization: HP2C Implementation

Full ED application implemented within the High Performance and High Productivity Computing Initiative (`www.hp2c.ch`)

- Symmetries (reduces complexity at cost of more communication)
- Supports multiple one- and two-dimensional lattices
- Multiple quantum models
  - Heisenberg
  - Fendley (computationally intensive, unlike SPIN benchmark)
  - FQHE (computationally intensive)

- OMP/MPI Implementation (simple loop-based directives)

```
// local offdiagonal part
for (unsigned j = 0; j < asubspace.size(); ++j) {
    :
  index_type lastk = bspace.last(j);
  typename bspace_type::biterator it = bspace.begin(j);
#pragma omp parallel for
  for (index_type k = 0; k < lastk; ++k) {
    if (it[k].size == 0) continue;
    state_type state = it[k].state | a_state;
    value_type nv =
     matrix.def.b_apply(state, pvec, *this, ae, it[k].size);
    vspace.slice(rvec, j)[k] += nv;
  }
}
```

Cray and PGI C++ compiler optimizations issues

Cray and PGI C++ compiler optimizations issues

On Cray XE6/XK6 expect CCE
to generate adequate code

Cray and PGI C++ compiler optimizations issues

On Cray XE6/XK6 expect CCE to generate adequate code

- CCE 8.0.2 vs. GNU 4.6.2 performance comparison

Cray and PGI C++ compiler optimizations issues

On Cray XE6/XK6 expect CCE
to generate adequate code

- CCE 8.0.2 vs. GNU 4.6.2
  performance comparison
- Various thread-core pinnings

Cray and PGI C++ compiler optimizations issues

On Cray XE6/XK6 expect CCE to generate adequate code

- CCE 8.0.2 vs. GNU 4.6.2 performance comparison
- Various thread-core pinnings

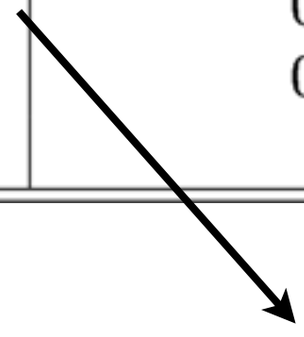
Threads	Pinning (-cc)	Cray	GNU
4	0,1,2,3	1011	667
4	0,2,4,6	1635	592
8	cpu	960	433

Cray and PGI C++ compiler optimizations issues

On Cray XE6/XK6 expect CCE to generate adequate code

- CCE 8.0.2 vs. GNU 4.6.2 performance comparison
- Various thread-core pinnings

Threads	Pinning (-cc)	Cray	GNU
4	0,1,2,3	1011	667
4	0,2,4,6	1635	592
8	cpu	960	433



Cray and PGI C++ compiler optimizations issues

On Cray XE6/XK6 expect CCE to generate adequate code

- CCE 8.0.2 vs. GNU 4.6.2 performance comparison
- Various thread-core pinnings

Threads	Pinning (-cc)	Cray	GNU
4	0,1,2,3	1011	667
4	0,2,4,6	1635	592
8	cpu	960	433

PAPI Event	Cray	GNU
PAPI_TOT_CYC	2'900'762'613'118	1'512'844'977'806
PAPI_FPU_IDL	1'130'511'978'612	242'206'647'131
PAPI_L2_TCM	1'422'885'720	74'482'188

Cray and PGI C++ compiler optimizations issues

On Cray XE6/XK6 expect CCE to generate adequate code

- CCE 8.0.2 vs. GNU 4.6.2 performance comparison
- Various thread-core pinnings

Threads	Pinning (-cc)	Cray	GNU
4	0,1,2,3	1011	667
4	0,2,4,6	1635	592
8	cpu	960	433

PAPI Event	Cray	GNU
PAPI_TOT_CYC	2'900'762'613'118	1'512'844'977'806
PAPI_FPU_IDL	1'130'511'978'612	242'206'647'131
PAPI_L2_TCM	1'422'885'720	74'482'188

Cray and PGI C++ compiler optimizations issues

On Cray XE6/XK6 expect CCE to generate adequate code

- CCE 8.0.2 vs. GNU 4.6.2 performance comparison
- Various thread-core pinnings

Threads	Pinning (-cc)	Cray	GNU
4	0,1,2,3	1011	667
4	0,2,4,6	1635	592
8	cpu	960	433

PAPI Event	Cray	GNU
PAPI_TOT_CYC	2'900'762'613'118	1'512'844'977'806
PAPI_FPU_IDL	1'130'511'978'612	242'206'647'131
PAPI_L2_TCM	1'422'885'720	74'482'188

PAPI Event	Cray	GNU
PAPI_TOT_CYC	2'269'187'342'972	775'493'075'990
PAPI_FPU_IDL	1'311'686'430'846	141'553'650'048
PAPI_L2_TCM	821'641'494	80'416'142

Cray and PGI C++ compiler optimizations issues

On Cray XE6/XK6 expect CCE to generate adequate code

- CCE 8.0.2 vs. GNU 4.6.2 performance comparison
- Various thread-core pinnings

Vast performance difference!

Threads	Pinning (-cc)	Cray	GNU
4	0,1,2,3	1011	667
4	0,2,4,6	1635	592
8	cpu	960	433

PAPI Event	Cray	GNU
PAPI_TOT_CYC	2'900'762'613'118	1'512'844'977'806
PAPI_FPU_IDL	1'130'511'978'612	242'206'647'131
PAPI_L2_TCM	1'422'885'720	74'482'188

PAPI Event	Cray	GNU
PAPI_TOT_CYC	2'269'187'342'972	775'493'075'990
PAPI_FPU_IDL	1'311'686'430'846	141'553'650'048
PAPI_L2_TCM	821'641'494	80'416'142

Cray and PGI C++ compiler optimizations issues

On Cray XE6/XK6 expect CCE to generate adequate code

- CCE 8.0.2 vs. GNU 4.6.2 performance comparison
- Various thread-core pinnings

Vast performance difference!

- GNU uses full SSE width; unrolls loops, pipelines

Threads	Pinning (-cc)	Cray	GNU
4	0,1,2,3	1011	667
4	0,2,4,6	1635	592
8	cpu	960	433

PAPI Event	Cray	GNU
PAPI_TOT_CYC	2'900'762'613'118	1'512'844'977'806
PAPI_FPU_IDL	1'130'511'978'612	242'206'647'131
PAPI_L2_TCM	1'422'885'720	74'482'188

PAPI Event	Cray	GNU
PAPI_TOT_CYC	2'269'187'342'972	775'493'075'990
PAPI_FPU_IDL	1'311'686'430'846	141'553'650'048
PAPI_L2_TCM	821'641'494	80'416'142

Cray and PGI C++ compiler optimizations issues

On Cray XE6/XK6 expect CCE to generate adequate code

- CCE 8.0.2 vs. GNU 4.6.2 performance comparison
- Various thread-core pinnings

Vast performance difference!

- GNU uses full SSE width; unrolls loops, pipelines
- CCE does not

Threads	Pinning (-cc)	Cray	GNU
4	0,1,2,3	1011	667
4	0,2,4,6	1635	592
8	cpu	960	433

PAPI Event	Cray	GNU
PAPI_TOT_CYC	2'900'762'613'118	1'512'844'977'806
PAPI_FPU_IDL	1'130'511'978'612	242'206'647'131
PAPI_L2_TCM	1'422'885'720	74'482'188

PAPI Event	Cray	GNU
PAPI_TOT_CYC	2'269'187'342'972	775'493'075'990
PAPI_FPU_IDL	1'311'686'430'846	141'553'650'048
PAPI_L2_TCM	821'641'494	80'416'142

Cray and PGI C++ compiler optimizations issues

On Cray XE6/XK6 expect CCE to generate adequate code

- CCE 8.0.2 vs. GNU 4.6.2 performance comparison
- Various thread-core pinnings

Vast performance difference!

- GNU uses full SSE width; unrolls loops, pipelines
- CCE does not
- Cray bug reports filed, Cray has reproduced problem, working on optimizer

Threads	Pinning (-cc)	Cray	GNU
4	0,1,2,3	1011	667
4	0,2,4,6	1635	592
8	cpu	960	433

PAPI Event	Cray	GNU
PAPI_TOT_CYC	2'900'762'613'118	1'512'844'977'806
PAPI_FPU_IDL	1'130'511'978'612	242'206'647'131
PAPI_L2_TCM	1'422'885'720	74'482'188

PAPI Event	Cray	GNU
PAPI_TOT_CYC	2'269'187'342'972	775'493'075'990
PAPI_FPU_IDL	1'311'686'430'846	141'553'650'048
PAPI_L2_TCM	821'641'494	80'416'142

Optimal # threads / socket

Optimal # threads / socket

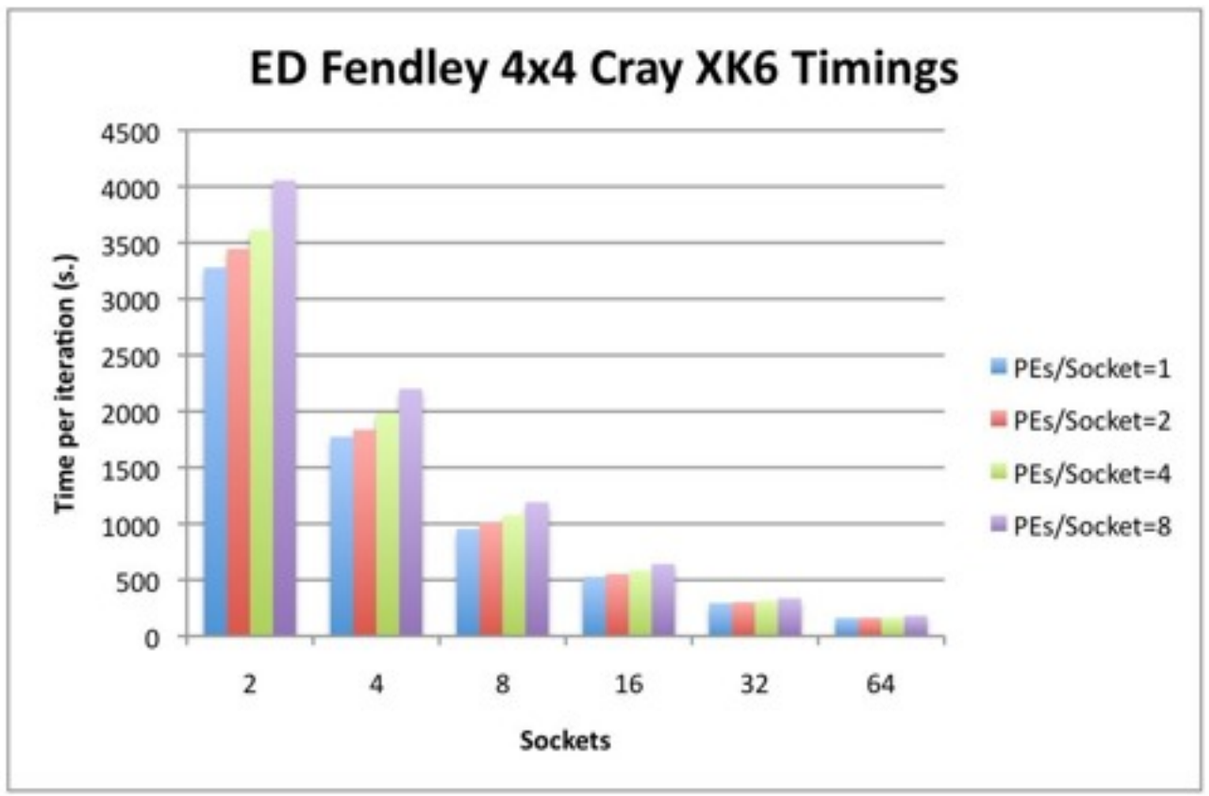
Performance comparison:

- Cray XK6: GNU 4.6.2
- SGI UV1000: Intel 11.1
- Various thread, pinning & socket configurations tried

Optimal # threads / socket

Performance comparison:

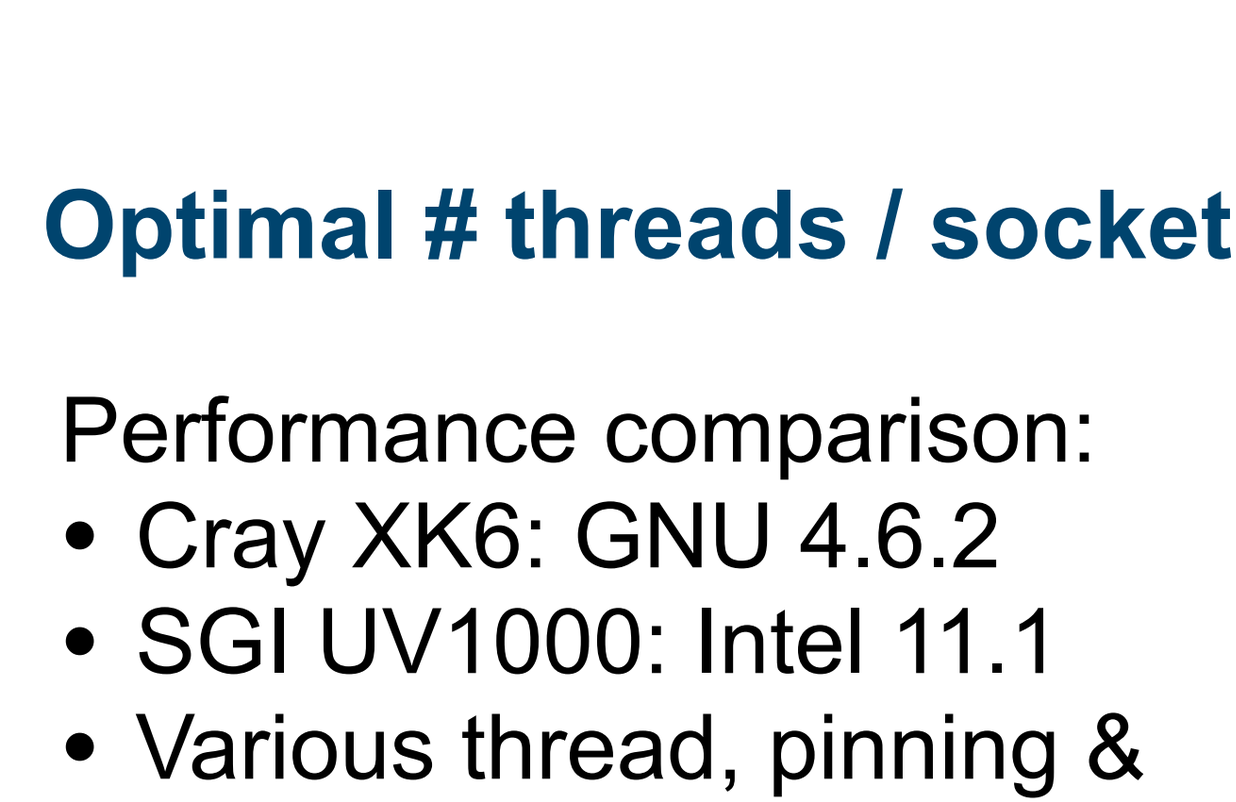
- Cray XK6: GNU 4.6.2
- SGI UV1000: Intel 11.1
- Various thread, pinning & socket configurations tried



Optimal # threads / socket

Performance comparison:

- Cray XK6: GNU 4.6.2
- SGI UV1000: Intel 11.1
- Various thread, pinning & socket configurations tried

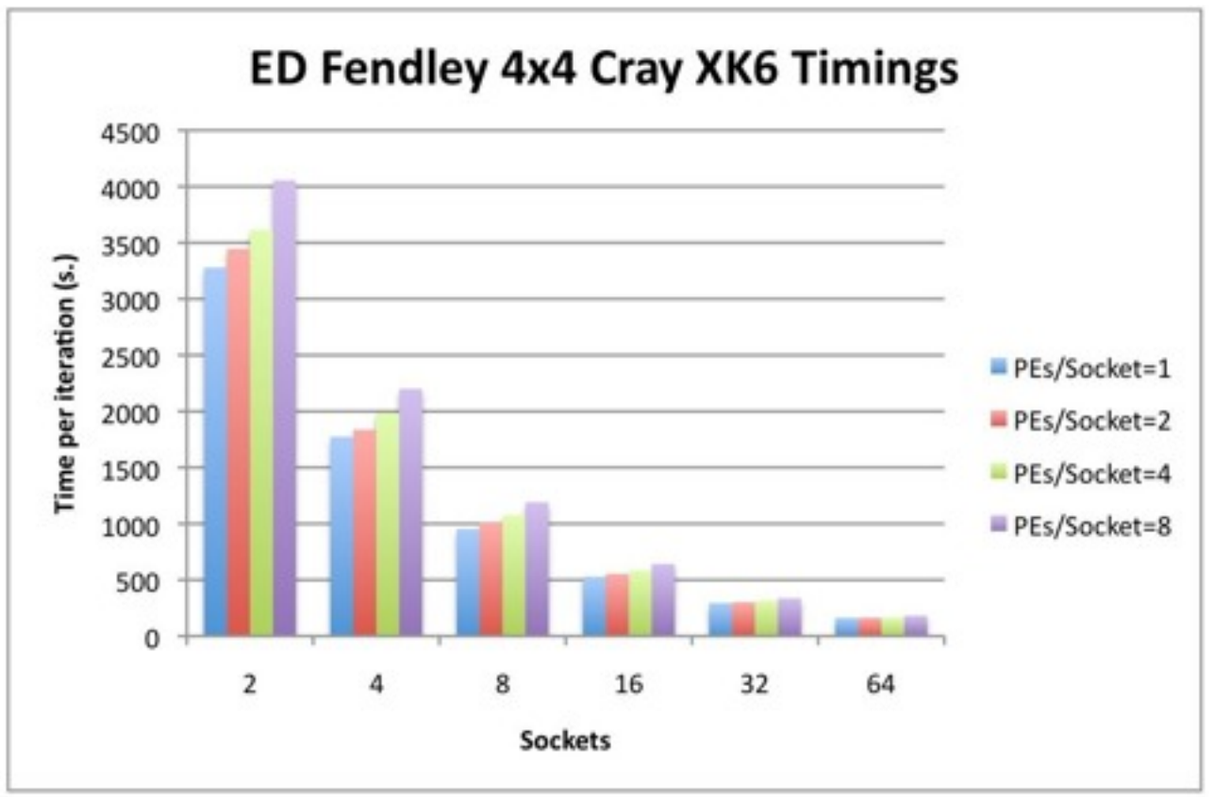


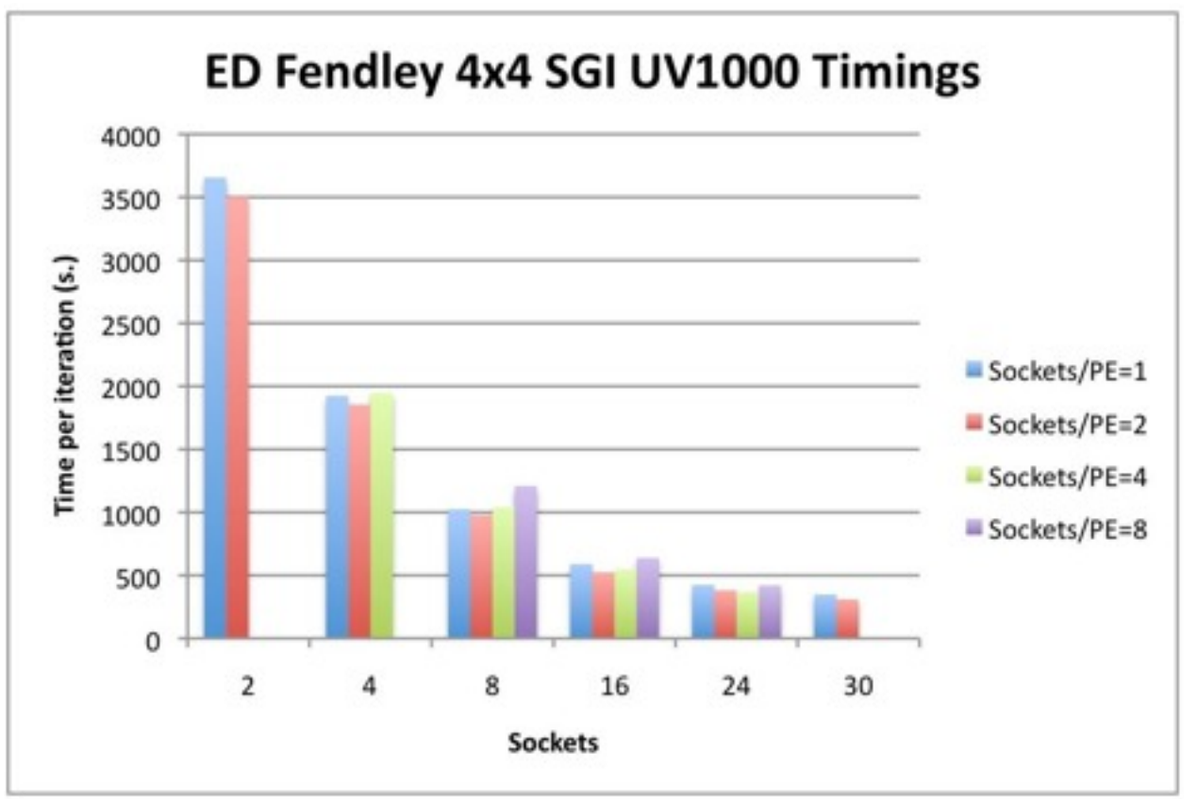
Optimal # threads / socket

Performance comparison:

- Cray XK6: GNU 4.6.2
- SGI UV1000: Intel 11.1
- Various thread, pinning & socket configurations tried

Conclusions:





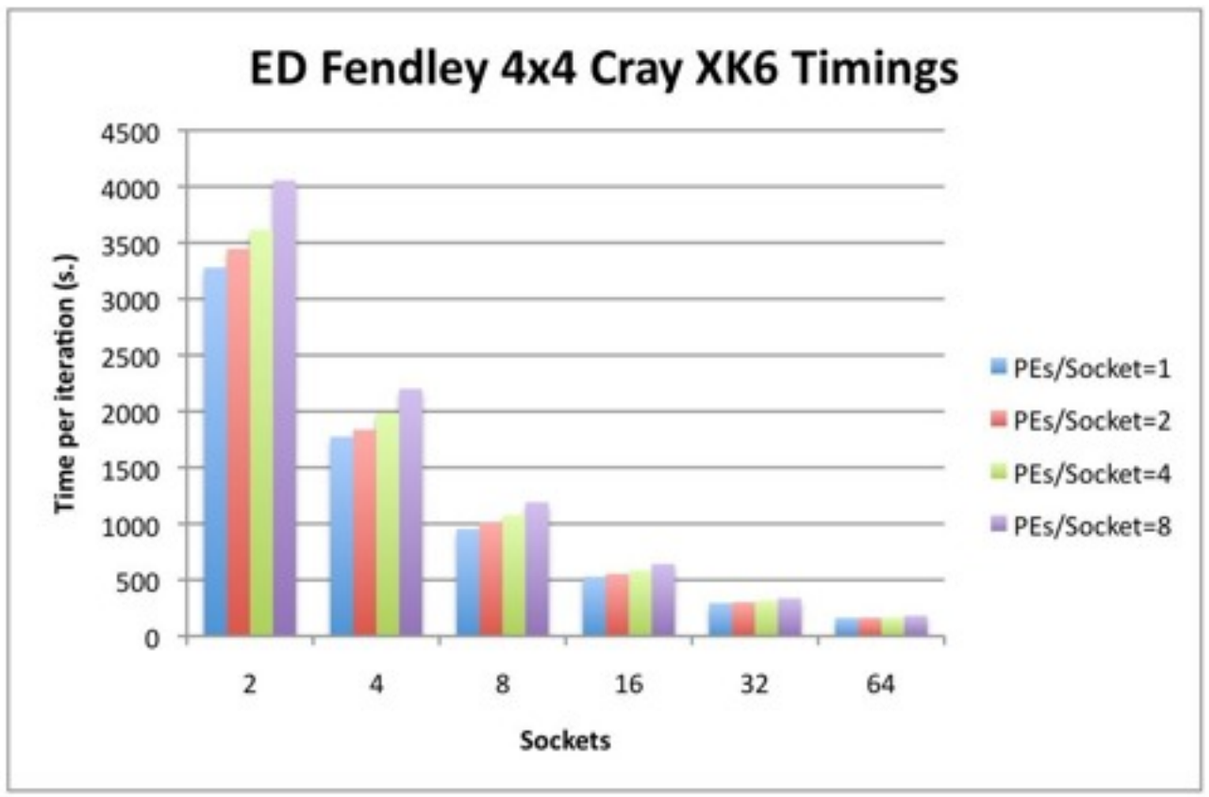
Optimal # threads / socket

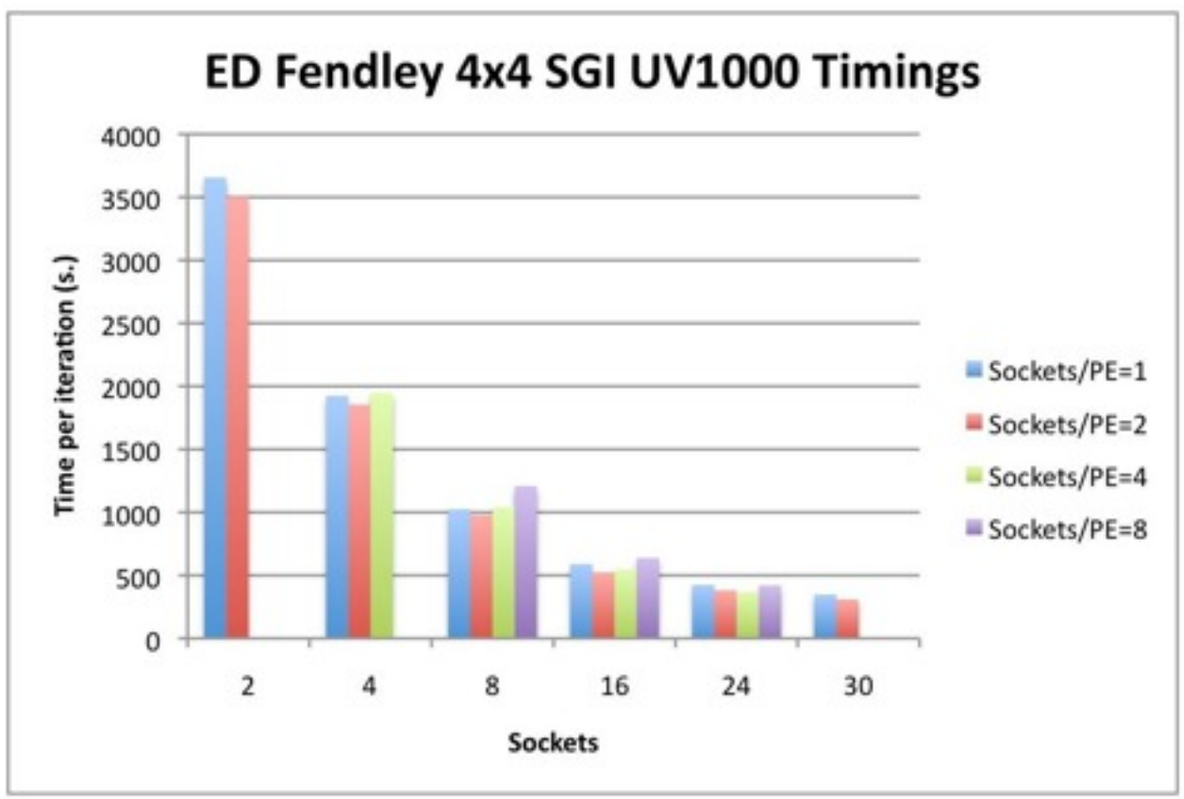
Performance comparison:

- Cray XK6: GNU 4.6.2
- SGI UV1000: Intel 11.1
- Various thread, pinning & socket configurations tried

Conclusions:

- Cray XK6: 1 MPI process / socket, 16 threads





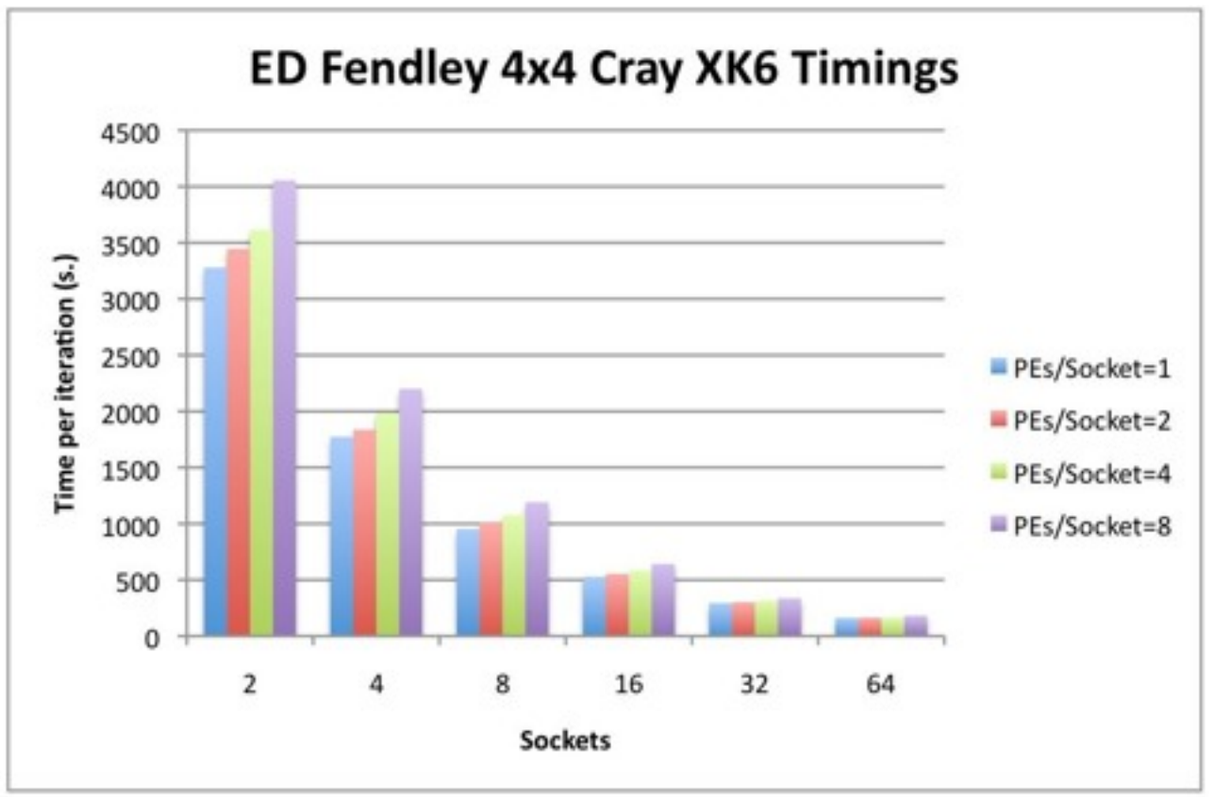
Optimal # threads / socket

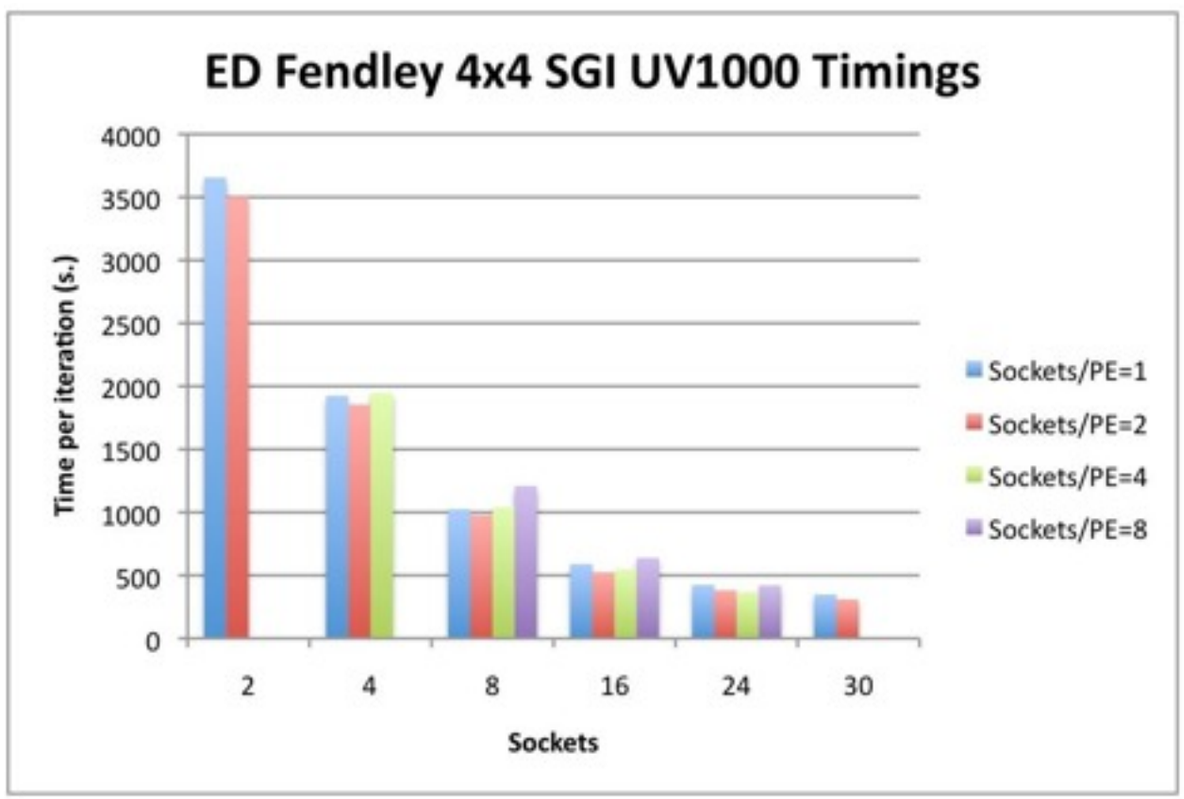
Performance comparison:

- Cray XK6: GNU 4.6.2
- SGI UV1000: Intel 11.1
- Various thread, pinning & socket configurations tried

Conclusions:

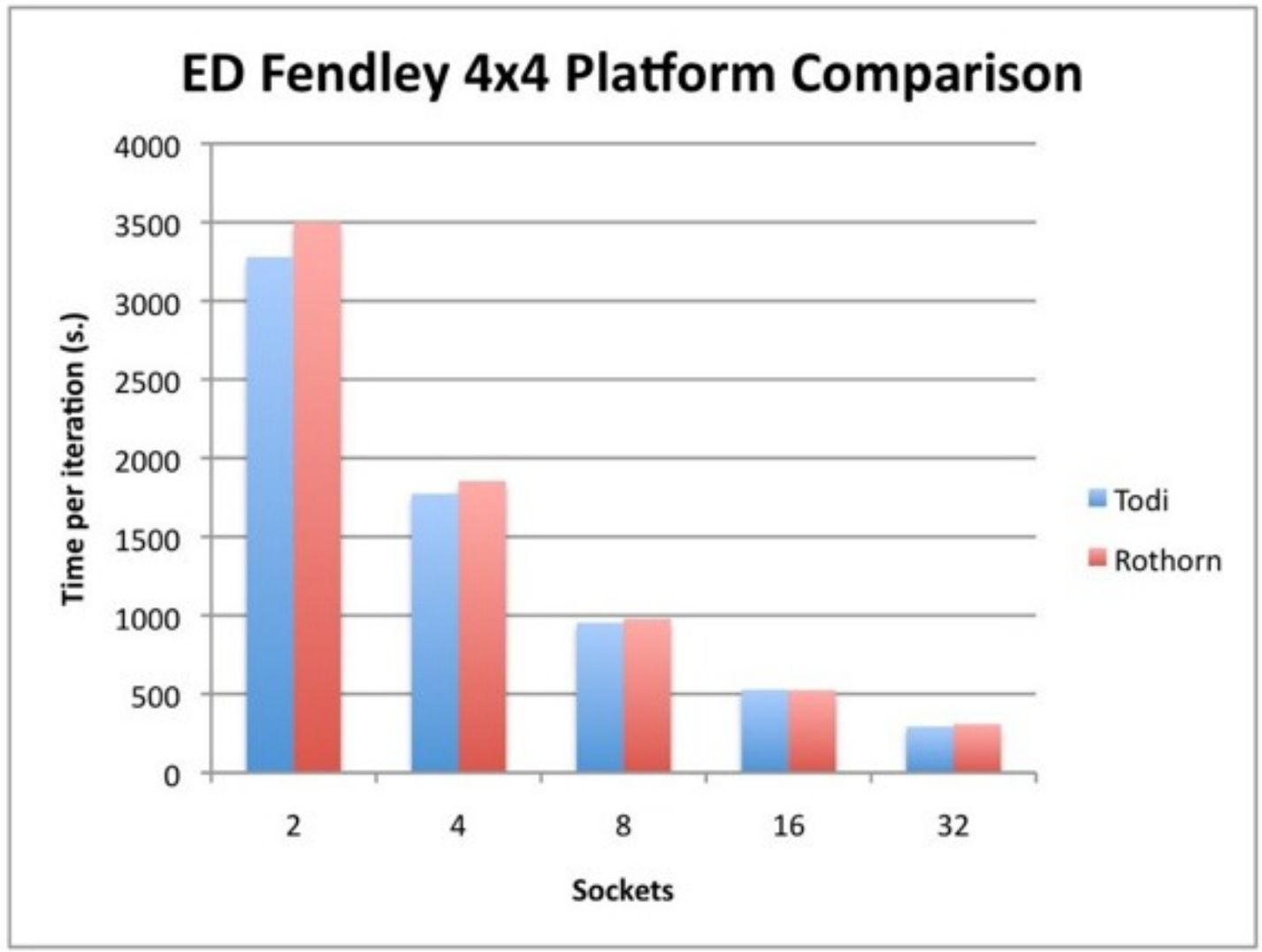
- Cray XK6: 1 MPI process / socket, 16 threads
- SGI UV1000: 2 sockets / MPI Process, 16 threads





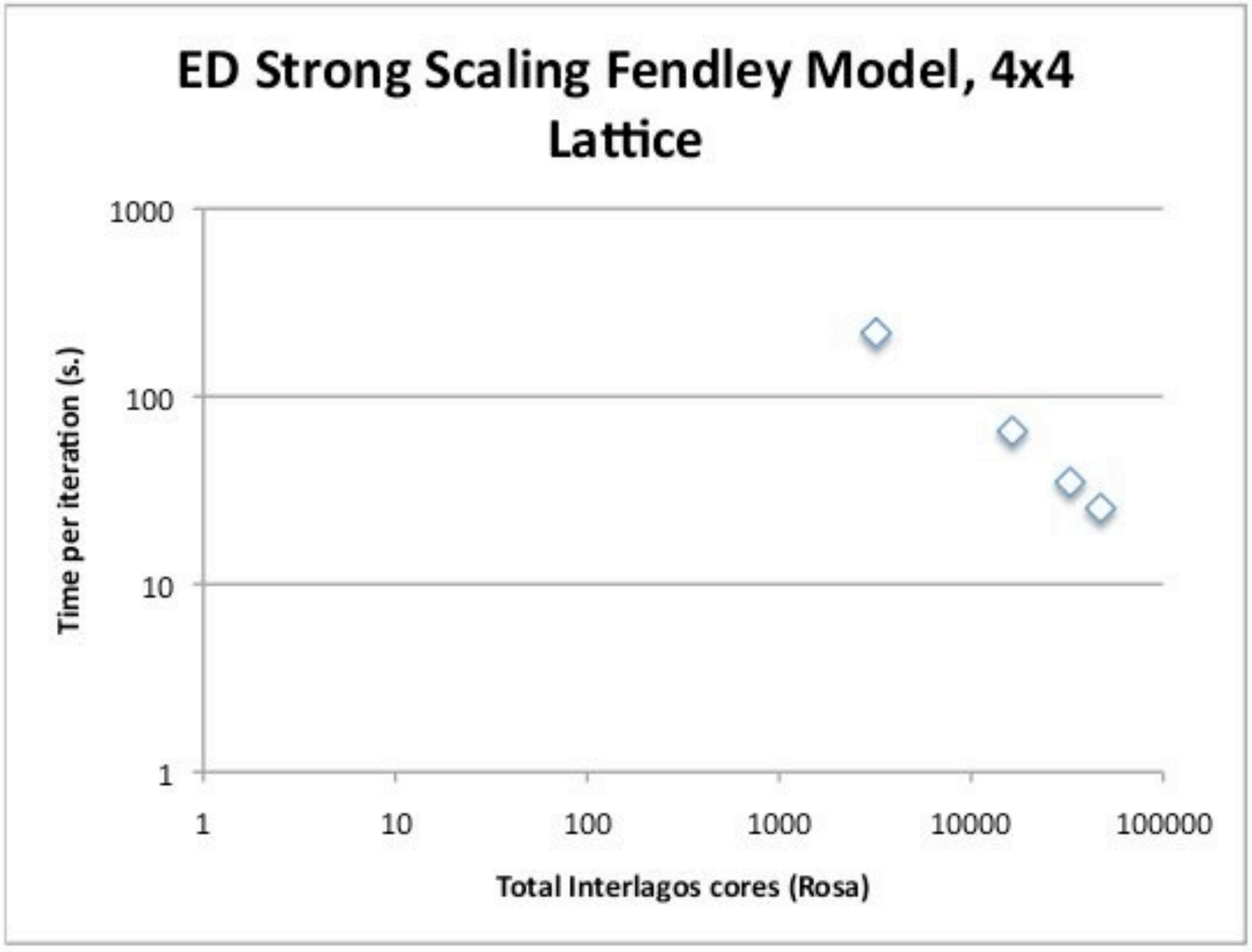
Performance Comparison Cray XE6 / SGI UV1000

- Cray XE6: AMD Interlagos, Gemini, GNU 4.6.2
- SGI UV1000: Intel Westmere, NUMAlink, Intel 11.1

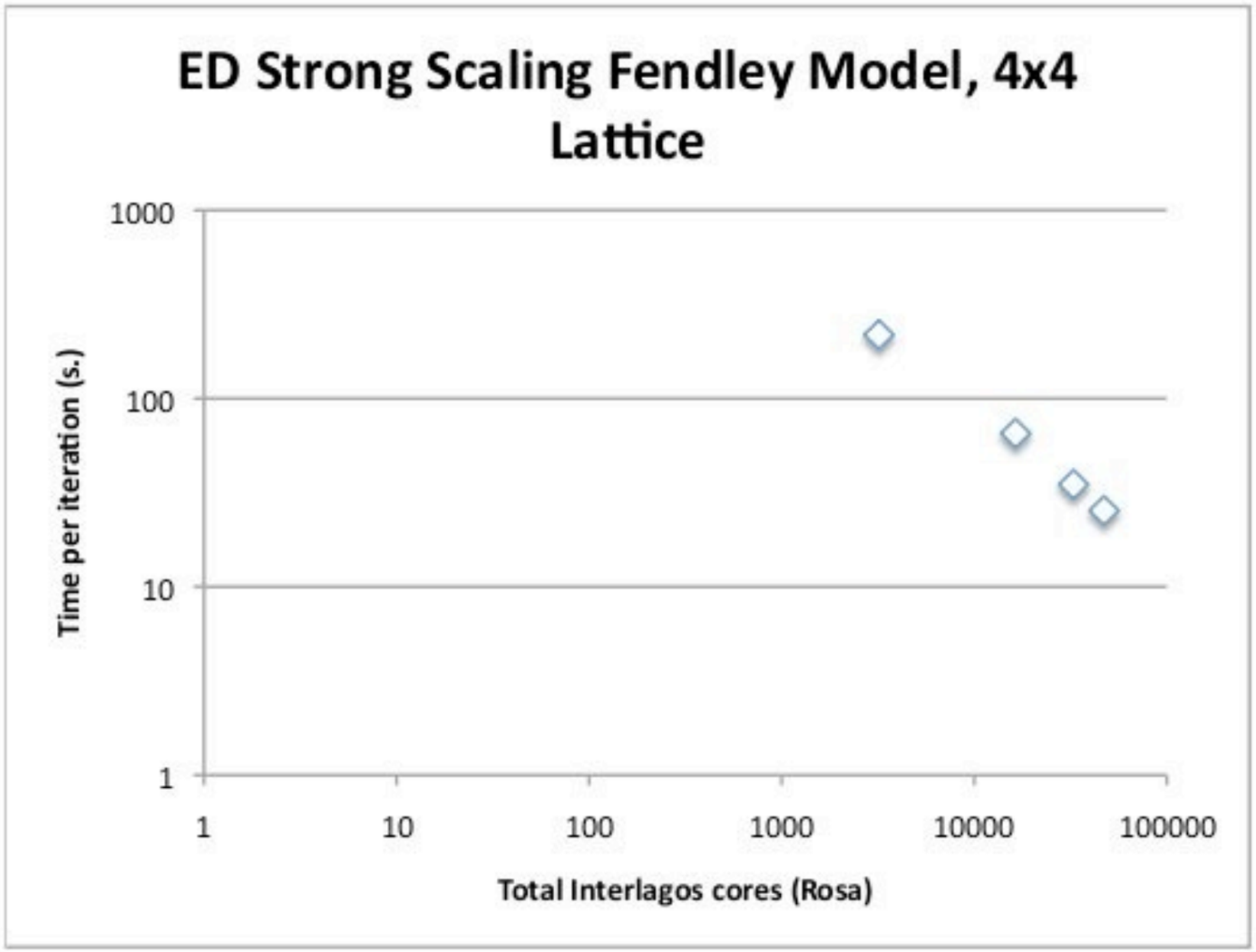


ED performance, scientifically relevant test case (4x4 lattice, Fendley model, no symmetries)

ED performance, scientifically relevant test case (4x4 lattice, Fendley model, no symmetries)



ED performance, scientifically relevant test case (4x4 lattice, Fendley model, no symmetries)



MPI Processes	Total Cores	Time/iteration (s.)
100	3200	218
512	16384	65.4
1024	32768	35
1476	47232	25.4

Take-home messages

Take-home messages

- SPIN benchmark on single-node:
  - **Intel Sandybridge:** best core/socket/node performance, but hyperthreading brings little improvement
  - **AMD Interlagos:** tricky to optimize thread placement, Open64 best single-node performance
  - **Intel Westmere:** competitive with Interlagos

Take-home messages

- SPIN benchmark on single-node:
  - **Intel Sandybridge:** best core/socket/node performance, but hyperthreading brings little improvement
  - **AMD Interlagos:** tricky to optimize thread placement, Open64 best single-node performance
  - **Intel Westmere:** competitive with Interlagos
- Cray Gemini: MPI two-sided is competitive with all other message passing paradigms, slight improvements with UPC

Take-home messages

- SPIN benchmark on single-node:
  - **Intel Sandybridge:** best core/socket/node performance, but hyperthreading brings little improvement
  - **AMD Interlagos:** tricky to optimize thread placement, Open64 best single-node performance
  - **Intel Westmere:** competitive with Interlagos
- Cray Gemini: MPI two-sided is competitive with all other message passing paradigms, slight improvements with UPC
- ED: Cray and PGI C++ compilers generate inferior code compared to GNU (bug reports filed)

Take-home messages

- SPIN benchmark on single-node:
  - **Intel Sandybridge:** best core/socket/node performance, but hyperthreading brings little improvement
  - **AMD Interlagos:** tricky to optimize thread placement, Open64 best single-node performance
  - **Intel Westmere:** competitive with Interlagos
- Cray Gemini: MPI two-sided is competitive with all other message passing paradigms, slight improvements with UPC
- ED: Cray and PGI C++ compilers generate inferior code compared to GNU (bug reports filed)
- ED nearest-neighbor feature ensures scaling to large configurations

Take-home messages

- SPIN benchmark on single-node:
  - **Intel Sandybridge:** best core/socket/node performance, but hyperthreading brings little improvement
  - **AMD Interlagos:** tricky to optimize thread placement, Open64 best single-node performance
  - **Intel Westmere:** competitive with Interlagos
- Cray Gemini: MPI two-sided is competitive with all other message passing paradigms, slight improvements with UPC
- ED: Cray and PGI C++ compilers generate inferior code compared to GNU (bug reports filed)
- ED nearest-neighbor feature ensures scaling to large configurations
- Hybrid OMP/MPI viable even for simplest model

Take-home messages

- SPIN benchmark on single-node:
  - **Intel Sandybridge:** best core/socket/node performance, but hyperthreading brings little improvement
  - **AMD Interlagos:** tricky to optimize thread placement, Open64 best single-node performance
  - **Intel Westmere:** competitive with Interlagos
- Cray Gemini: MPI two-sided is competitive with all other message passing paradigms, slight improvements with UPC
- ED: Cray and PGI C++ compilers generate inferior code compared to GNU (bug reports filed)
- ED nearest-neighbor feature ensures scaling to large configurations
- Hybrid OMP/MPI viable even for simplest model

Thank you for your attention!

wsawyer@cscs.ch