SAND2010-0573C

Performance Concerns for Coupling Hybrid-Parallel Kernels

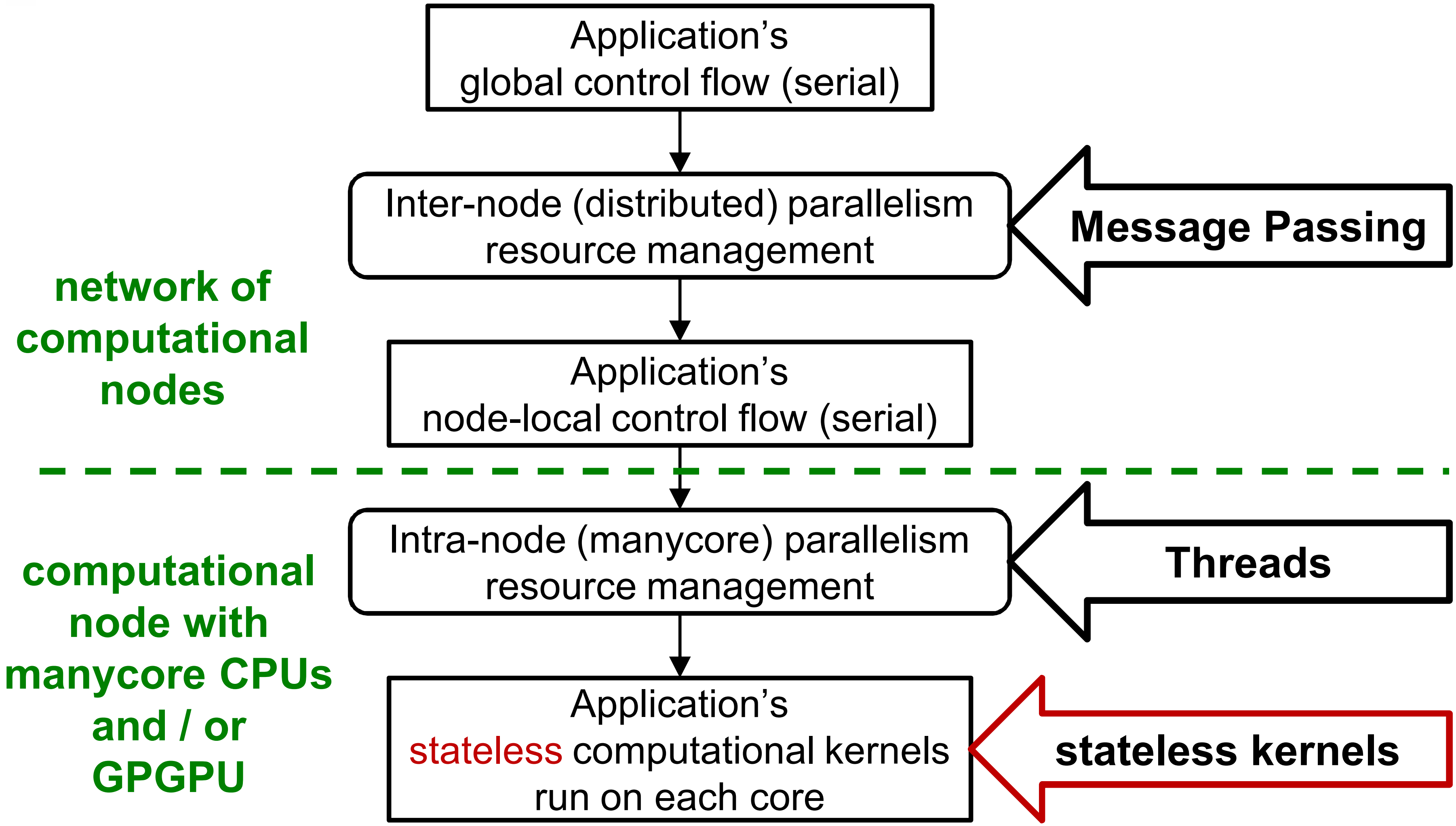
H. Carter Edwards
Sandia National Laboratory

SIAM Conference on
Parallel Processing for Scientific Computing
February 23-26, 2010

SAND 2010-

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy's National Nuclear Security Administration
under contract DE-AC04-94AL85000.

A Model for Hybrid Parallelism: Layered Separation of Concerns



Thread Parallelism (nested within message passing parallelism)

- **Threads running on multicore / manycore hardware**
  - Multiple cores per CPU socket
  - Multiple CPU sockets per distributed memory node
  - Multiple GPGPU devices per distributed memory node
- **Threads contend for memory resources**
  - Cores within a CPU socket contend for access to main memory
  - Cores within a CPU socket contend for shared cache memory
  - Cores contend for the node's pool of main memory
  - GPGPU devices have their own memory
- **Separate concerns**
  - Thread-parallel control and memory contention
  - Application's computational work

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Thread-Parallel Control and its Programming Model

- **Existing thread-parallel mechanisms**
  - Pthreads: library-based standard, *does not* define a model
  - Intel TBB: C++ STL-like hiding of Pthreads, defines a model
  - OpenMP: compiler-based standard, defines a model
  - CUDA: language+library for NVIDIA GPGPUs, defines a model
  - others …
- **Trilinos' ThreadPool library**
  - trilinos.sandia.gov
  - Simple and minimalistic library layered on Pthreads
  - Defines a model conceptually compatible with TBB and CUDA
  - Simple C-language application programmer interface (API)

ThreadPool API : Run Threads

```
TPI_Run_threads( & my_subprogram, & my_work_info, 0);
```

- **my_subprogram is called once on each thread**
  - ‘my_work_info’ is passed to, and shared by, each call
  - It is the application’s ‘struct’ or ‘class’ of work information

```
void my_subprogram( TPI_Work * work )
{
  work->info  /* = & my_work_info */
  work->count /* = number of threads */
  work->rank  /* = rank of this thread */
  /* computations over ‘rank’ of ‘count’ work */
}
```

ThreadPool API : Run Work

```
TPI_Run( &my_subprogram, &my_work_info, work_count, 0);
```

- **my_subprogram is called by threads**
  - Called ‘work_count’ number of times
  - Threads perform work-stealing with a work queue
  - Approximate load balancing for work_count >> # threads

```
void my_subprogram( TPI_Work * work )
{
  work->info  /* = & my_work_info */
  work->count /* = work_count */
  work->rank  /* = rank of this call */
  /* computations over ‘rank’ of ‘count’ work */
}
```

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy’s National Nuclear Security Administration
under contract DE-AC04-94AL85000.

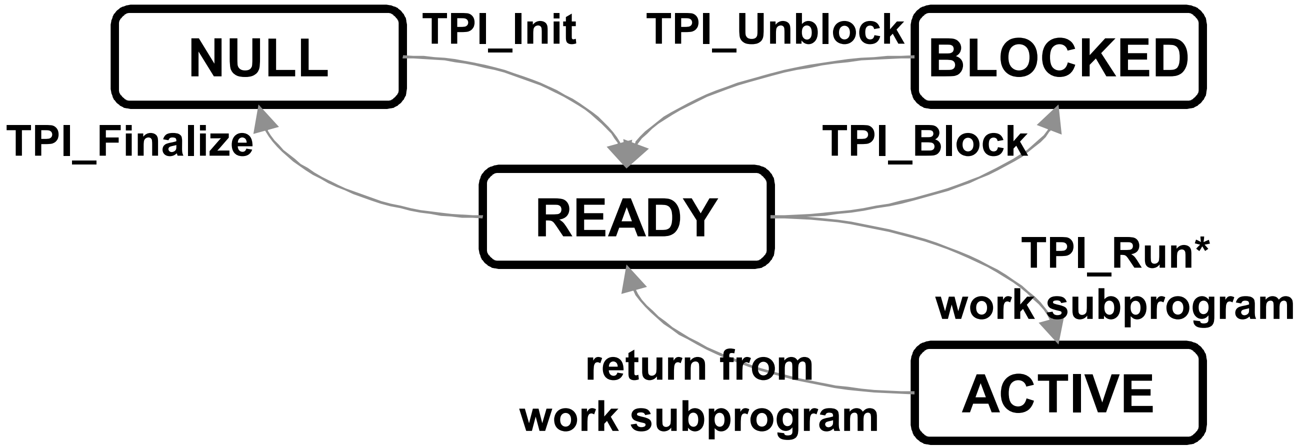
ThreadPool Performance Concerns

- **Thread creation overhead**
  - ➢ **Create threads once and maintain them within a pool**

- **Work initiation overhead** (call to TPI_Run*)
  - **Attach work subprogram to threads, initiate work**
  - **Large overhead to activate blocked threads**
  - **However, unblocked "spinning" threads compete with external threads (OpenMP, TBB) for resources**
  - ➢ **Explicitly manage the state of the ThreadPool threads**

- **Thread affinity to cores, sockets, and memory**
  - **Inter-socket threads cannot share cache**
  - **Sockets may have affinity to portions of memory**
  - ➢ **Non-uniform memory access (NUMA) controls**

ThreadPool State Diagram



- Usage model
  - An algorithm is comprised of a sequence of parallel kernels
    - E.g., a sparse matrix solve algorithm
  - Sequence of calls to TPI_Run* with these kernels
  - Threads are spinning in the READY state between calls
  - Threads can be blocked upon completion of the algorithm
  - Threads are unblocked upon initiation of the algorithm

ThreadPool API and Performance Concern: Work with a Reduction

- Kernels may require a reduction; e.g., dot product
  - Requires update of a shared reduction variable
  - Locking serializes parallel execution
  - ➢ Use fan-in pattern to minimize serialization

```
TPI_Run_threads_reduce(
  & my_subprogram , & my_work_info ,
  & my_reduce_join , & my_reduce_init ,
  sizeof(my_reduce_data) , & my_reduce_data );
```

- my_reduce_data: Reduction result
- my_reduce_join: Function to reduce pairs of intermediate results
- my_reduce_init: Function to initialize intermediate results

ThreadPool API : Work with a Reduction

- Example: reduction functions for a dot product
  - Each thread computes a partial sum for its work
  - Each thread has exclusive access to a partial sum variable
  - Partial sum variables initialized via initialize function
  - Partial sum variables reduced via join function

```
void dsum_reduce_join( TPI_Work * work, const void * rhs )
{ *((double*) work->reduce) += *((const double *)rhs); }

void dsum_reduce_init( TPI_Work * work )
{ *((double*) work->reduce) = 0.0 ; }
```

- Deterministic calls to reduction functions
  - No race conditions as with locking
  - Repeatable results, independent of thread completion order

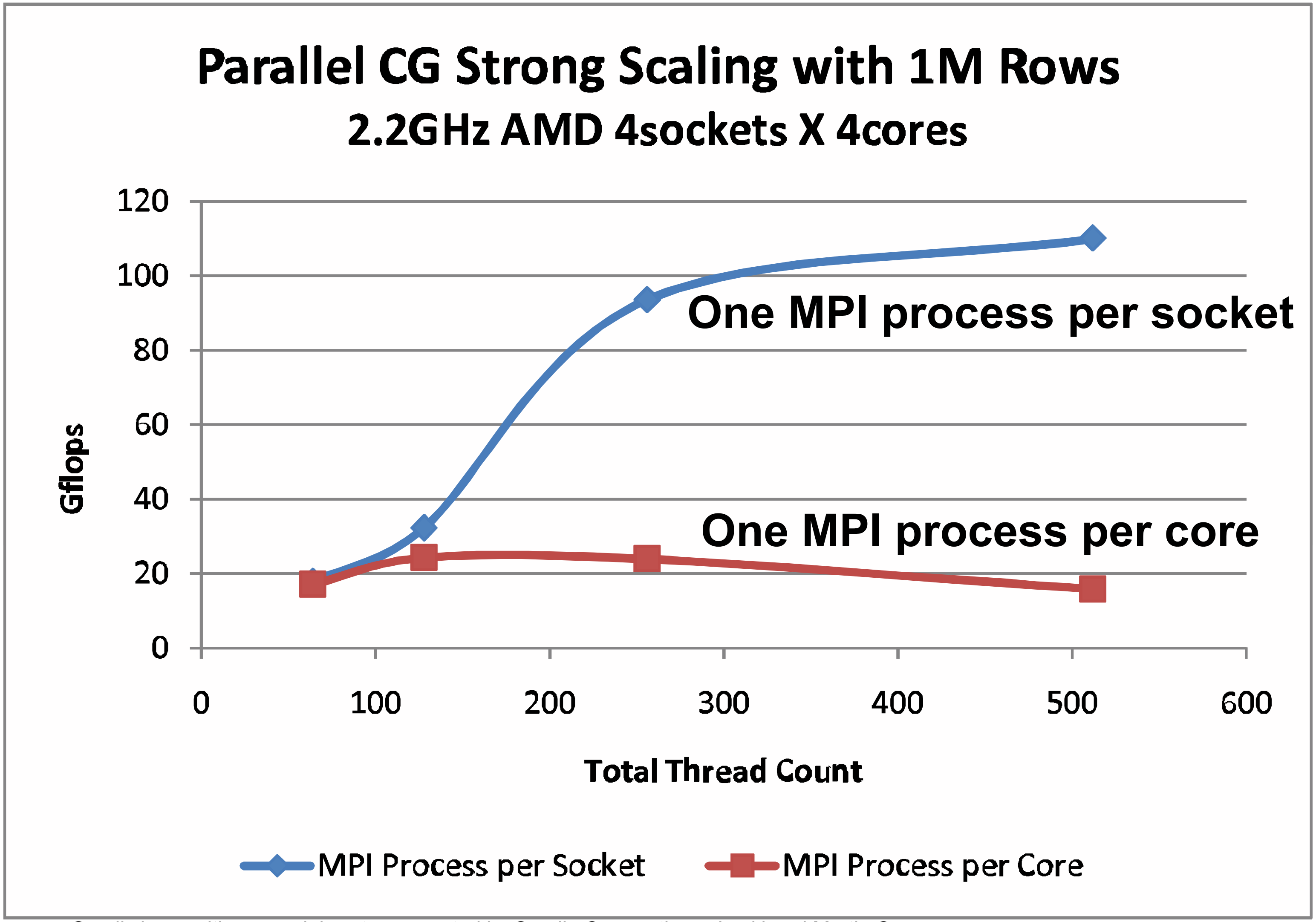
Performance Study: Conjugate Gradient Sparse Matrix Solve

- SANDIA Laboratories' GLORY cluster
  - 272 quad-socket nodes with 2.2 GHz AMD quad-core CPUs
  - Sockets have non-uniform memory access (NUMA)
  - Infiniband interconnect

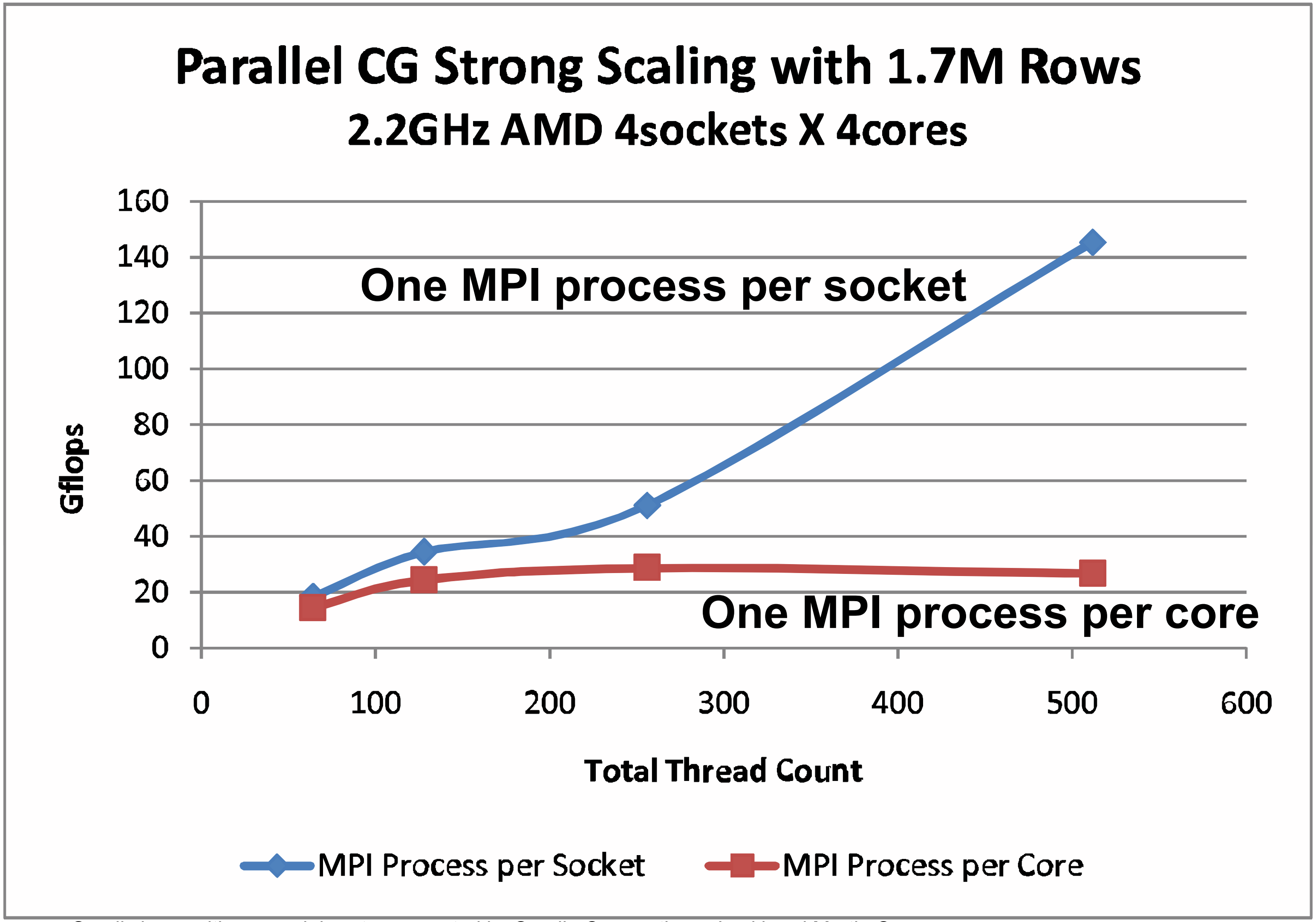
- Parallel execution modes:
  - One MPI process per core
  - One MPI process + three worker threads per socket
  - One MPI process + eleven worker threads per node

- Sparse matrix with ~27 non-zeros per row
  - 27point stencil on a cube
  - Cube partitioned via recursive coordinate bisection

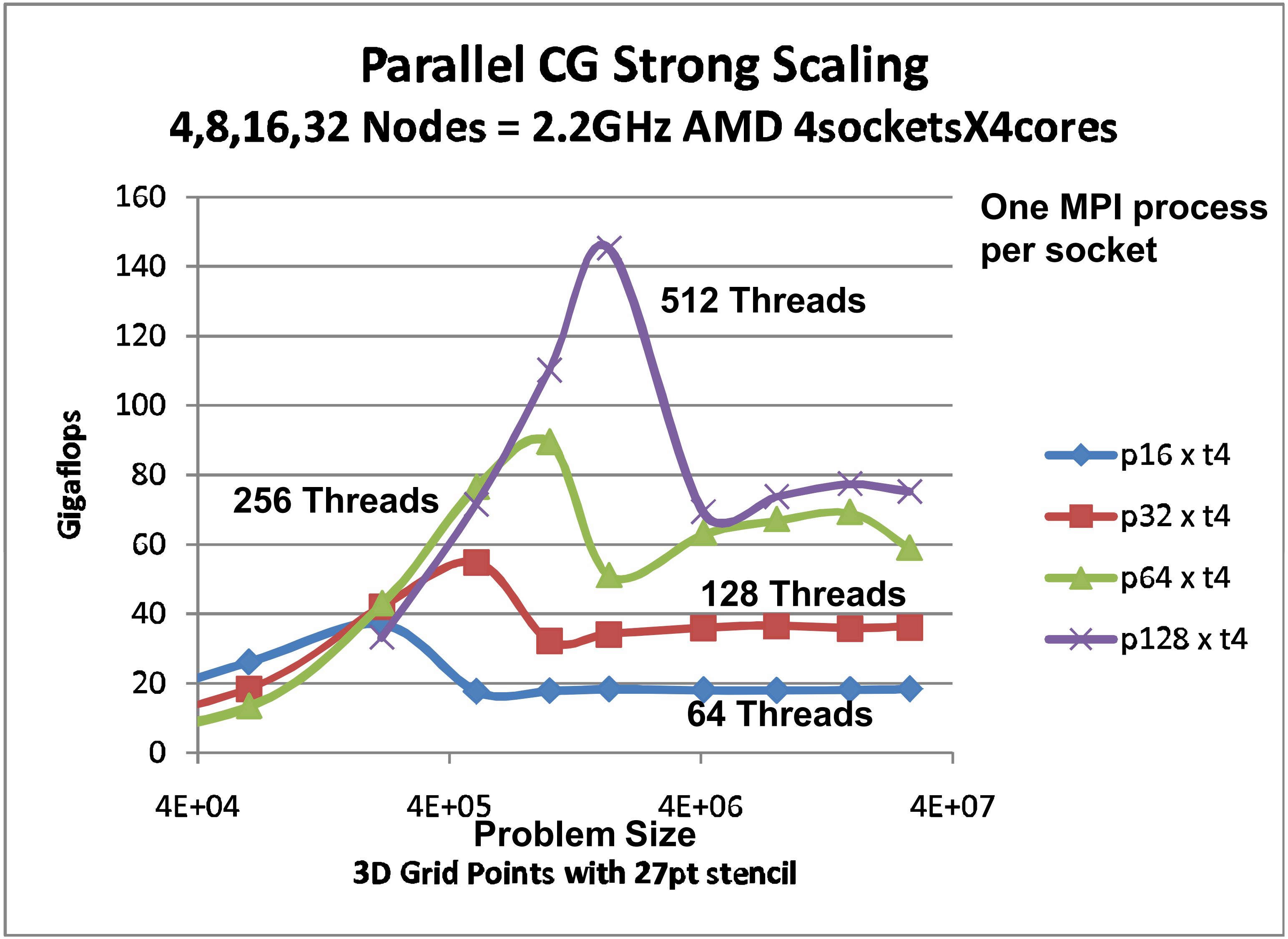
Performance Study: Conjugate Gradient Sparse Matrix Solve



Performance Study:
Conjugate Gradient Sparse Matrix Solve



Performance Study: Cache Effects



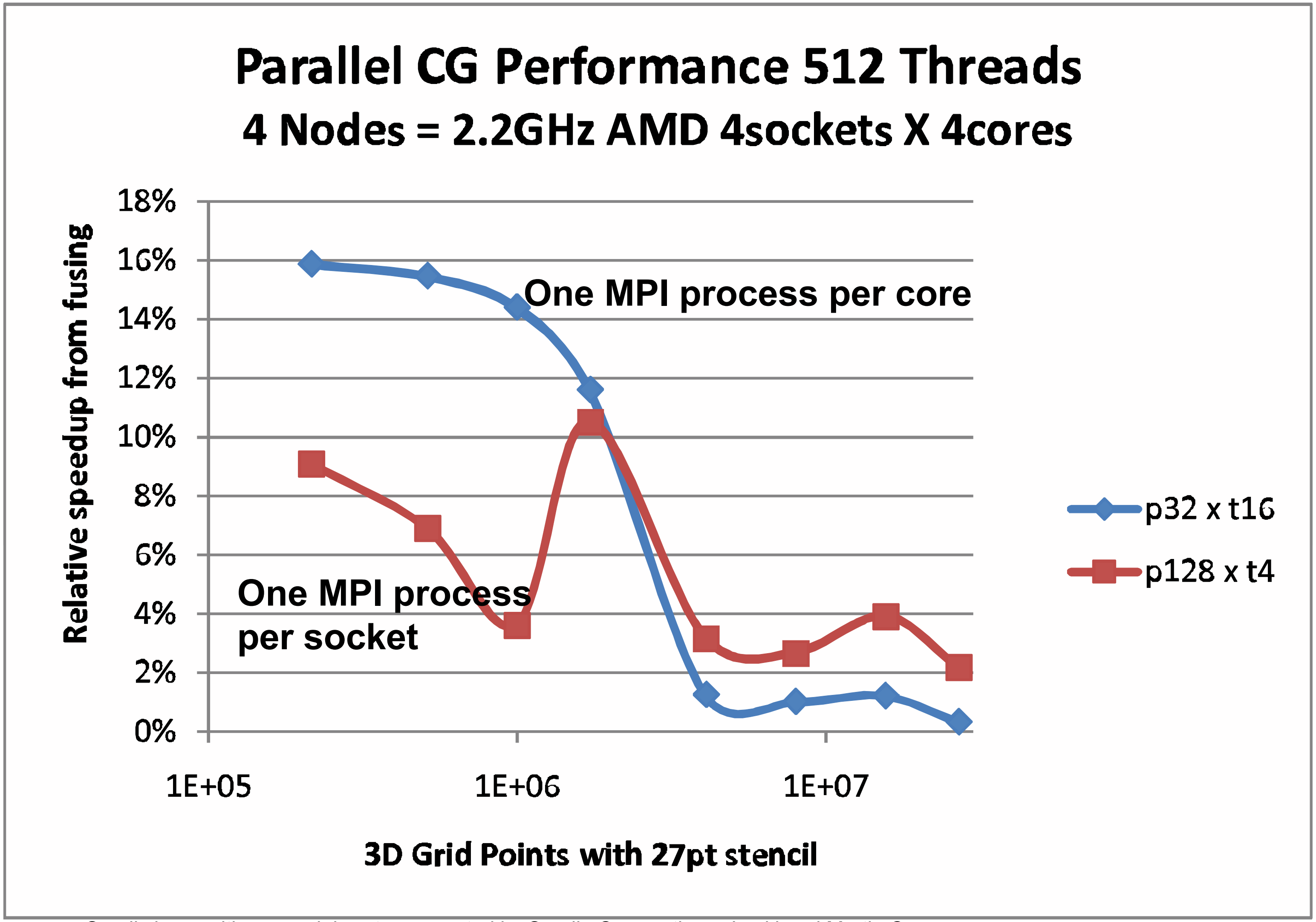
ThreadPool Performance : Fusing Kernels

- Each call to TPI_Run* is a synchronization
  - Introduces overhead unless a reduction is required
  - ➢ Remove unnecessary synchronization by fusing kernels
  - Example: conjugate gradient solver vector update

```
X = X + α * P ;    /* vector update */
R = R - α * P ;    /* vector update */
δ = dot( R , R ); /* reduction */
```

  - Fuse these three kernels into a single kernel

Performance Study:
CG Solve Speedup from Fusing Kernels



Summary

- **Hybrid MPI / thread parallel programming**
  - Can provide dramatic performance gains
  - Fusing kernels can further improve performance
  - Thread pool programming model
    - Conceptually conformal with OpenMP, TBB, CUDA, …
    - Blocked vs. ready "spinning" threads impacts performance

- **Trilinos' ThreadPool library**
  - Simple, minimalistic C language interface
  - Explicit control over blocked vs. ready "spinning" state
  - trilinos.sandia.gov

Summary:
Layered Separation of Concerns