Shared Memory Extensions for MPI

Sandia National Laboratories

Proposed API

- MPI_COMM_ALLOC_MEM(comm, size, info, baseptr)
  - IN comm input communicator (handle)
  - IN size size of memory segment in bytes (non-negative int)
  - IN info info argument (handle)
  - OUT baseptr pointer to beginning of memory segment allocated (choice)

- MPI_COMM_FREE_MEM(comm, base)
  - IN comm input communicator (handle)
  - IN base initial address of memory segment allocated by MPI_COMM_ALLOC_MEM (choice)

Proposed Semantics

- MPI_COMM_ALLOC_MEM()
  - Collective call
  - Allocates region of shared memory accessible by ranks in input communicator
  - No guarantee of identical baseptr across ranks
  - Otherwise, semantics are same as MPI_ALLOC_MEM()
  - Returns MPI_ERR_COMM if no shared memory is possible
  - Return MPI_ERR_NO_MEM if memory is exhausted
- MPI_COMM_FREE_MEM()
  - Collective call
  - Same semantics as MPI_FREE_MEM()

Sandia National Laboratories

Why shared memory?

- Performance
  - Direct load/store access between processes is more efficient than any MPI communication method
- Ease of use
  - Supports structured programming
    - Data is private until explicitly shared
  - Easier to use than threads
    - Where everything is shared and must be explicitly made private
- Reduce replicated state across processes
- Available on all systems (that I know of)

Sandia National Laboratories

Why do this in MPI? (1/2)

- Performance
  - Integrating into MPI offers opportunity for optimization
    - POSIX shared memory allocation is not collective
    - Making it collective offers opportunity to optimize for layout and access
    - Also can make message passing more efficient
      - Affinity for multi-rail transfers
    - Potentially useful for integrating accelerators
    - May optimize checkpointing/resiliency
      - No need to replicate shared memory for all ranks
  - Opportunity for using non-POSIX shared memory portably

Why do this in MPI? (2/2)

- Integration with MPI run-time system
  - Simplifies shared memory allocation
    - An MPI application would want run-time system information to allocate shared memory anyway
  - Simplifies shared memory cleanup
    - Leftover state on node ends up being MPI's fault anyway ☺
- Allows integration with MPI tools
  - Debuggers, performance debuggers, etc.
- Ease of programming
  - Incremental approach for existing MPI applications
  - POSIX shared memory is not easy to use
- Ease of implementation
  - MPI implementations already use shared memory

Hybrid MPI/Multi-Threaded Programming in Scientific Computing

Workshop to Explore the Introduction of Threads into SNL-ASC codes

August 30, 2010

***Michael Wolf*, Mike Heroux, Erik Boman**
Scalable Algorithms Department (1416)
Extreme-scale Algorithms and Software Institute (EASI)

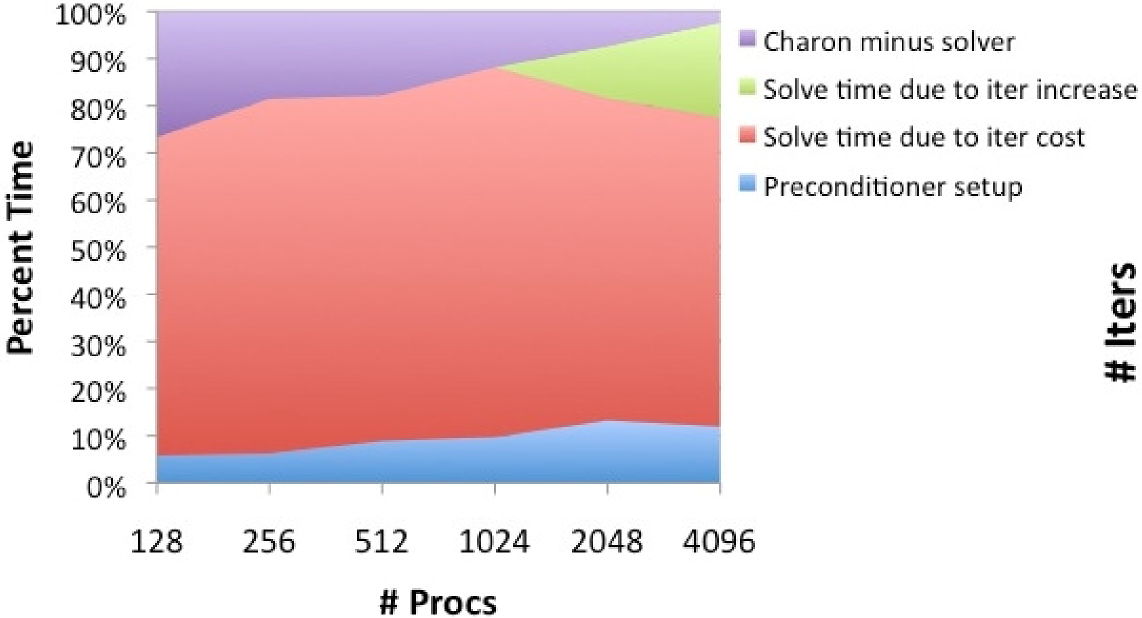
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

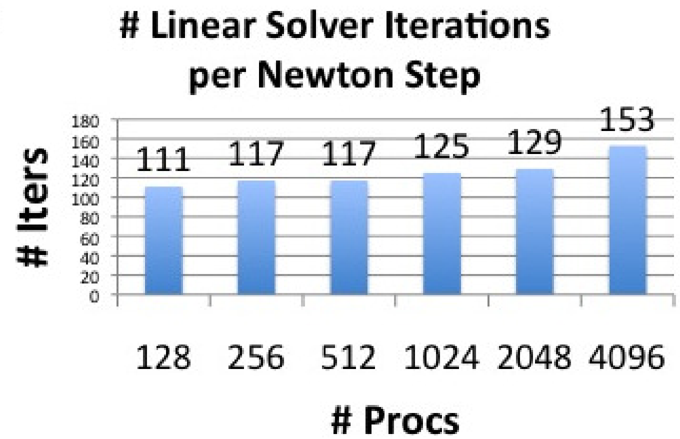
Outline

- Bimodal MPI-only/MPI + X programming
  - Integrating hybrid kernels into MPI-only applications in painless manner
  - MPI extensions for shared memory allocation
  - Simple example
  - Work in progress: Hybrid MPI/multithreaded PCG

Sandia National Laboratories

Motivation

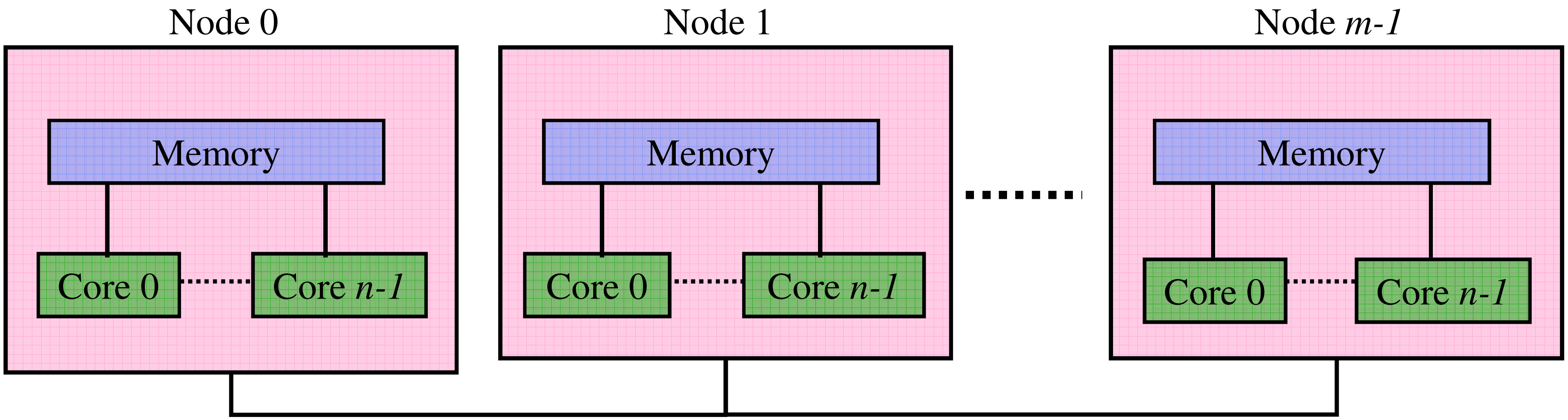




Strong scaling of Charon on TLCC (P. Lin, J. Shadid 2009)

- Domain decomposition preconditioning with incomplete factorizations
- Inflation in iteration count due to number of subdomains
- With scalable threaded triangular solves
  - Solve triangular system on larger subdomains
  - Reduce number of subdomains (MPI tasks)

MPI + Hybrid MPI/Multithreaded Programming



- Parallel machine with $p = m * n$ processors:
  - m = number of nodes
  - n = number of shared memory cores per node
- Two typical ways to program
  - Way 1: p MPI processes (flat MPI-only)
  - Way 2: m MPI processes with n threads per MPI process
- Third way (bimodal approach)
  - "Way 1" in some parts of the execution (the app)
  - "Way 2" in others (the solver)

Sandia National Laboratories

MPI Shared Memory Allocation

Idea:

- Shared memory alloc/free functions:
  - MPI_Comm_alloc_mem
  - MPI_Comm_free_mem
- Status:
  - Available in current development branch of OpenMPI
  - Demonstrated usage with threaded triangular solve

Collaborators: B. Barrett, R. Brightwell - SNL; Vallee, Koenig - ORNL

Simple MPI Program

```
double *x = new double[4];
double *y = new double[4];

MPIkernel1(x,y);
MPIkernel2(x,y);

delete [] x;
delete [] y;
```

- Simple MPI application
  - Two distributed memory/MPI kernels
- Want to replace an MPI kernel with more efficient hybrid MPI/threaded
  - Threading on multicore node

Simple MPI + Hybrid Program

```
double *x = new double[4];
double *y = new double[4];

MPIkernel1(x,y);
MPIkernel2(x,y);

delete [] x;
delete [] y;
```

```
MPI_Comm_size(MPI_COMM_NODE, &nodeSize);
MPI_Comm_rank(MPI_COMM_NODE, &nodeRank);

double *x, *y;

MPI_Comm_alloc_mem(MPI_COMM_NODE,n*nodeSize*sizeof(double),
.                              MPI_INFO_NULL, &x);
MPI_Comm_alloc_mem(MPI_COMM_NODE,n*nodeSize*sizeof(double),
.                              MPI_INFO_NULL, &y);

MPIkernel1(&(x[nodeRank * n]),&(y[nodeRank * n]));

if(nodeRank==0)
{
.    hybridKernel2(x,y);
}

MPI_Comm_free_mem(MPI_COMM_NODE, &x);
MPI_Comm_free_mem(MPI_COMM_NODE, &y);
```

n=4

- Very minor changes to code
  - MPIKernel1 does not change
- Hybrid MPI/Threaded kernel runs on rank 0 of each node
  - Threading on multicore node

Iterative Approach to Hybrid Parallelism

- Many sections of parallel applications scale extremely well using MPI-only model.
  - Don't change these sections much
- Approach allows introduction of multithreaded kernels in iterative fashion
  - "Tune" how multithreaded an application is
- Can focus on parts of application that don't scale with MPI-only programming
- Approach requires few changes to MPI-only sections

Iterative Approach to Hybrid Parallelism

```
MPI_Comm_size(MPI_COMM_NODE, &nodeSize);
MPI_Comm_rank(MPI_COMM_NODE, &nodeRank);

double *x, *y;

MPI_Comm_alloc_mem(MPI_COMM_NODE,n*nodeSize*sizeof(double),
.                         MPI_INFO_NULL, &x);
MPI_Comm_alloc_mem(MPI_COMM_NODE,n*nodeSize*sizeof(double),
.                         MPI_INFO_NULL, &y);

MPIkernel1(&(x[nodeRank * n]),&(y[nodeRank * n]));

if(nodeRank==0)
{
.    hybridKernel2(x,y);
}

MPI_Comm_free_mem(MPI_COMM_NODE, &x);
MPI_Comm_free_mem(MPI_COMM_NODE, &y);
```

- Can use 1 hybrid kernel

Iterative Approach to Hybrid Parallelism

```
MPI_Comm_size(MPI_COMM_NODE, &nodeSize);
MPI_Comm_rank(MPI_COMM_NODE, &nodeRank);

double *x, *y;

MPI_Comm_alloc_mem(MPI_COMM_NODE,n*nodeSize*sizeof(double),
.                                MPI_INFO_NULL, &x);
MPI_Comm_alloc_mem(MPI_COMM_NODE,n*nodeSize*sizeof(double),
.                                MPI_INFO_NULL, &y);

if(nodeRank==0)
{
.    hybridKernel1(x,y);
.    hybridKernel2(x,y);
}

MPI_Comm_free_mem(MPI_COMM_NODE, &x);
MPI_Comm_free_mem(MPI_COMM_NODE, &y);
```

- Or use 2 hybrid kernels

Sandia National Laboratories

Work in Progress
Bimodal MPI-only/Multithreaded PCG

PCG Algorithm

$$r_0 = b - Ax_0$$
$$z_0 = M^{-1} r_0$$
$$p_0 = z_0$$
$$\text{for } (k = 0;\ k < maxit,\ ||r_k|| < tol)$$
$$\{$$
$$.\qquad \alpha_k = \frac{r_k^T z_k}{p_k^T A p_k}$$
$$.\qquad x_{k+1} = x_k + \alpha_k p_k$$
$$.\qquad r_{k+1} = r_k - \alpha_k A p_k$$
$$.\qquad z_{k+1} = M^{-1} r_{k+1}$$
$$.\qquad \beta_k = \frac{r_{k+1}^T z_{k+1}}{r_k^T z_k}$$
$$.\qquad p_{k+1} = z_{k+1} + \beta_k p_k$$
$$\}$$

Used symmetric Gauss-Seidel as preconditioner (2 triangular solves)

Sandia National Laboratories

PCG Algorithm

$$r_0 = b - Ax_0$$

$$\boxed{z_0} = \boxed{M}^{-1}\boxed{r_0}$$

$$p_0 = z_0$$

for $(k = 0;\ k < maxit,\ ||r_k|| < tol)$

$\{$

- $\alpha_k = \frac{r_k^T z_k}{p_k^T A p_k}$
- $x_{k+1} = x_k + \alpha_k p_k$
- $r_{k+1} = r_k - \alpha_k A p_k$
- $\boxed{z_{k+1}} = \boxed{M}^{-1}\boxed{r_{k+1}}$
- $\beta_k = \frac{r_{k+1}^T z_{k+1}}{r_k^T z_k}$
- $p_{k+1} = z_{k+1} + \beta_k p_k$

$\}$

Shared memory variables

Sandia National Laboratories

PCG Algorithm – MPI part

$$\boxed{r_0 = b - Ax_0}$$

$$z_0 = M^{-1} r_0$$

$$\boxed{p_0 = z_0}$$

$$\text{for } (k = 0;\ k < maxit,\ ||r_k|| < tol)$$

$\{$

$$\boxed{\begin{aligned} \alpha_k &= \frac{r_k^T z_k}{p_k^T A p_k} \\ x_{k+1} &= x_k + \alpha_k p_k \\ r_{k+1} &= r_k - \alpha_k A p_k \end{aligned}}$$

$$z_{k+1} = M^{-1} r_{k+1}$$

$$\boxed{\begin{aligned} \beta_k &= \frac{r_{k+1}^T z_{k+1}}{r_k^T z_k} \\ p_{k+1} &= z_{k+1} + \beta_k p_k \end{aligned}}$$

$\}$

MPI-only operations

Sandia National Laboratories

PCG Algorithm – Threaded Part

$$r_0 = b - Ax_0$$

$$\boxed{z_0 = M^{-1} r_0}$$

$$p_0 = z_0$$

$$\text{for } (k = 0;\ k < maxit,\ ||r_k|| < tol)$$

$$\{$$

$$.\qquad \alpha_k = \frac{r_k^T z_k}{p_k^T A p_k}$$

$$.\qquad x_{k+1} = x_k + \alpha_k p_k$$

$$.\qquad r_{k+1} = r_k - \alpha_k A p_k$$

$$.\qquad \boxed{z_{k+1} = M^{-1} r_{k+1}}$$

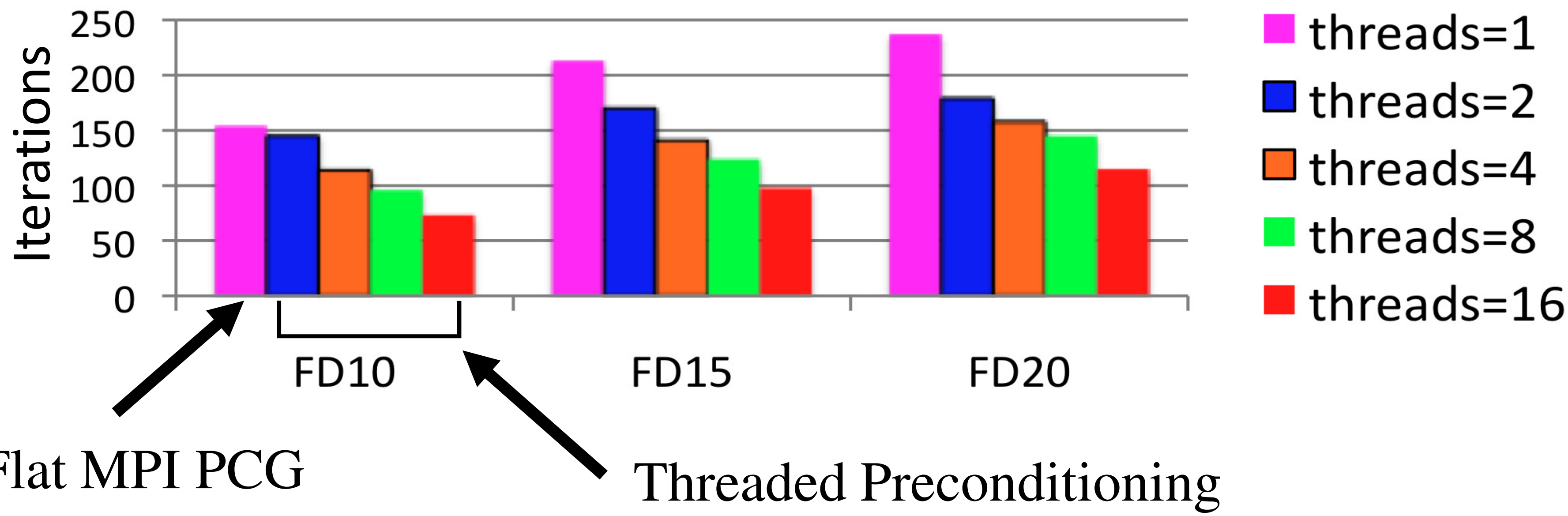
$$.\qquad \beta_k = \frac{r_{k+1}^T z_{k+1}}{r_k^T z_k}$$

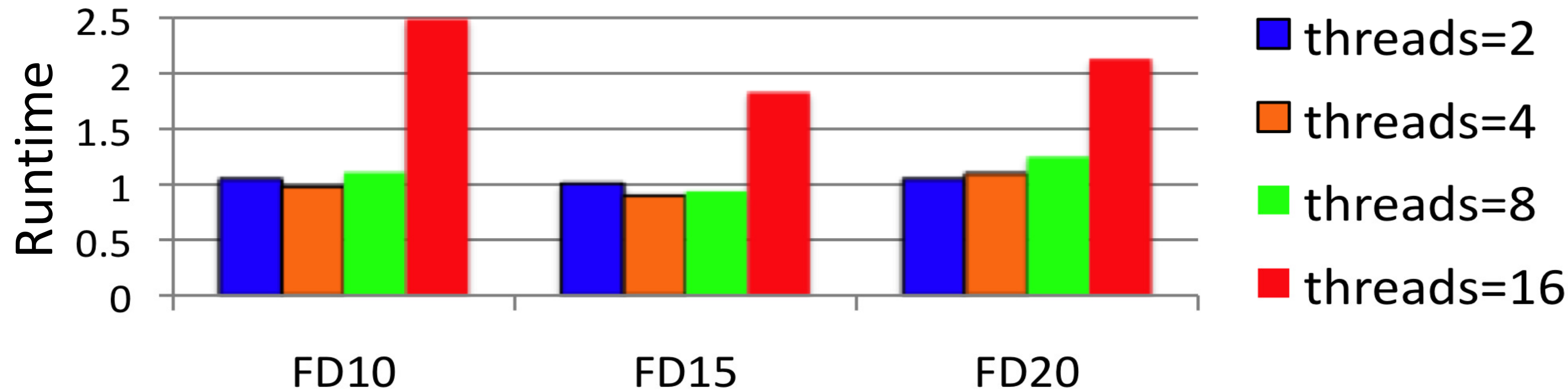
$$.\qquad p_{k+1} = z_{k+1} + \beta_k p_k$$

$$\}$$

Multithreaded block preconditioning to reduce number of subdomains

Preliminary PCG Results





Runtime relative to flat MPI PCG

Sandia National Laboratories

Summary: Bimodal MPI-only / MPI + X Programming

- Interface traditional MPI-only applications with efficient MPI + X kernels
  - Only change parts of applications that don't scale
- MPI shared memory allocation useful
  - Allows seamless combination of traditional MPI programming with MPI+X kernels
- Iterative approach to multithreading
- Implemented PCG using MPI shared memory extensions and level set method
  - Effective in reducing iterations
  - Runtime did not scale (work in progress)
  - Better triangular solver algorithms needed