A Computational Mathematician's Guide to High Performance Computing

Shane Sawyer

Research Associate, Joint Institute for Computational Sciences
University of Tennessee, Knoxville

shane-sawyer@tennessee.edu

February 18, 2015

Acknowledgements

Special thanks to colleagues who provided valuable feedback:
Dr. Vince Betro, Dr. Travis Thompson, Dr. Ryan Glasby,
and Dr. Taylor Erwin.

Overview of talk

1. **Introduction** What is high performance computing
2. **Programming Models** Threads, Processes, Distributed/Shared Memory
3. **Hardware** Multicore processors, memory hierarchies, accelerators
4. **Libraries** Existing libraries that simplify development and deliver performance
5. **Optimization** When to optimize, available tools
6. **Taking the Next Step** Where to find additional resources

Introduction

High performance computing is ubiquitous.

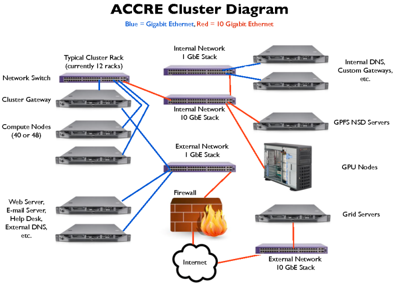
Titan supercomputer at ORNL. Image courtesy of Oak Ridge National Laboratory.

- Broadly defined: it is a collection of resources that offer more performance than desktops.
- Computational tasks that are too large (memory/operations) for a single resource.

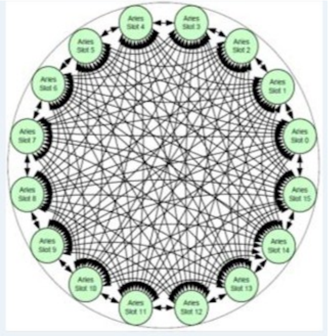
Clusters vs. Supercomputers

- **Clusters**
    - Collection of resources (servers, desktops, ...)
    - Interconnect (ethernet, InfiniBand, ...)
    - Commodity Linux distributions
- **Supercomputer**
    - Collection of specialized resources. Typically higher density.
    - High-speed interconnects. Higher performance networking topologies.
    - Customized compilers, tools, and OS.

Clusters vs. Supercomputers



Network diagram from ACCRE, Vanderbilt University.



Cray's Dragonfly topology. Image credit: Timothy Prickett Morgan , The Register.

Programming Models

Nomenclature

- **Threads** A single stream of execution.
- **Processes** Complete program with address space, code, I/O handles, ...
- **Shared Memory** Single pool of memory shared by resources. Explicit protection.
- **Distributed Memory** Memory is spread across resources. Explicit exchange of information.

Models

1. **OpenMP** An API, compiler directives, and runtime engine for shared memory, multithreading.
2. **MPI** A library and runtime for distributed memory parallel programming. Explicit message exchange.
3. **Hybrid** Use OpenMP on node and MPI between nodes for communication.
4. **Heterogeneous** Conventional resources with accelerators (GPU, Xeon Phi, FPGA, ...)

OpenMP Example

Dot product of two vectors.

```
#include <omp.h>
int main(void)
{
  double *a, *b, dotp;
  int i;
  // ... initialize and allocate a and b


  #pragma omp parallel for shared(a,b,N) \
    private(i) reduction(+ : dotp)
  for (i=0; i < N; ++i)
       dotp += a[i]*b[i];


  return 0;
}
```

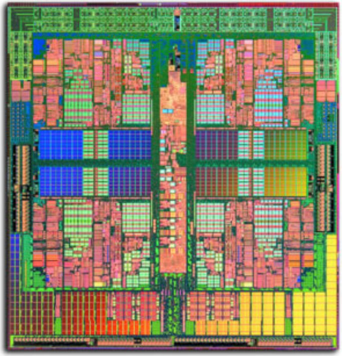
MPI Example

Dot product of two vectors.

```
#include <mpi.h>
int main(int argc, char* argv[])
{
  double *a, *b, dotp, temp;
  int i;
  // initialize MPI
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
  MPI_Comm_size(MPI_COMM_WORLD,&np);

  // ... initialize and allocate a and b, determine start and end
  dotp = temp = 0.;
  for ( i=START; i < END; ++i )
      temp += a[i] * b[i];

  MPI_Reduce(&temp,&dotp,1,MPI_DOUBLE,MPI_SUM,root,MPI_COMM_WORLD);

  MPI_Finalize();
  return 0;
}
```

Hardware

Multicore Processors are everywhere.



Quad-core AMD Opteron processor. Image credit: American Micro Devices, Inc.

- "The Free Lunch is Over" – Herb Sutter.
- Proliferation of multicore processors.
- Algorithms pushed towards parallelization.

Granularity

- **Socket** The physical packaging of cores with cache and interconnect.
- **Core** A complete processing element.
- **SIMD** Registers and execution units that allow one operation performed on multiple data in one tick.

NUMA and the Latency Hierarchy

NUMA – Non-Uniform Memory Access

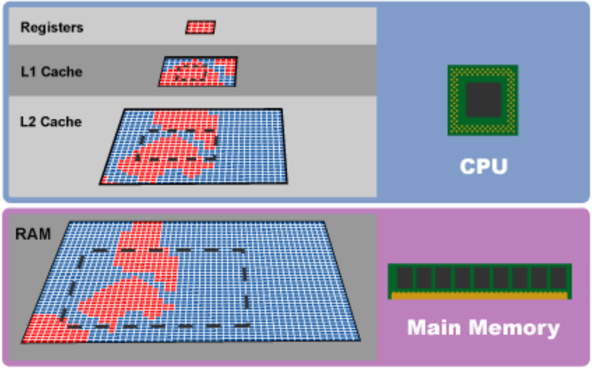


Image credit: Jon Stokes, ArsTechnica.

NUMA and the Latency Hierarchy

Memory Hierarchy

1. Registers – 1 cycle
2. Cache – L1 (4 cycles) → L2 (10 cycles) → L3 (40-75 cycles)
3. RAM – 100ns
4. Disk – 2ms

Data Locality

- Goal: Maintain high FP intensity
- Algorithms: Reuse data in cache
- Reflected in many modern linear algebra packages
- Example: create a tiling of matrices for multiplication.

Matrix Tiling

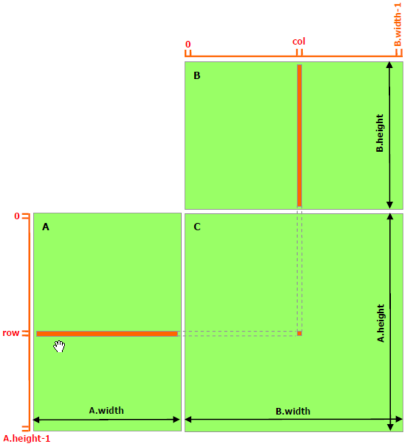


Image credit: Nvidia.

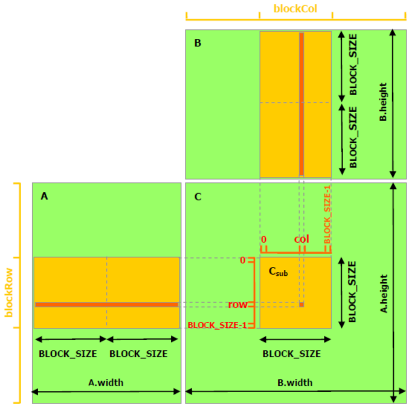


Image credit: Nvidia.

Libraries

Libraries for Computational Mathematics

- Parallel programming is challenging – changing technology, effort spent on low-level details
- Using libraries allows quicker development time, less debugging, abstracts communication details, lets application writers focus on their problem
- Plethora of excellent libraries for computational mathematics: linear algebra, nonlinear solvers, graph partitioning, PDEs, ...

Linear Algebra

- Modern libraries take advantage of hardware advances: ATLAS, Intel Math Kernel Library (MKL), PLASMA (ICL), blaze-lib
- Libraries for Heterogenous systems:
    - Intel Xeon Phi: MKL (with automatic offload support), MAGMA-MIC (ICL)
    - Nvidia GPU: CUBLAS, MAGMA (ICL)
- Parallel libraries: PETSc, Trilinos
- PDEs: deal.II, FEniCS, libMesh

blaze-lib: CG Example

```cpp
const size_t NN( N*N );

blaze::CompressedMatrix<double,rowMajor> A( NN, NN );
blaze::DynamicVector<double,columnVector> x( NN, 1.0 ), b( NN, 0.0 ),
                                          r( NN ), p( NN ), Ap( NN );
double alpha, beta, delta;

// ... Initializing the sparse matrix A

// Performing the CG algorithm
r = b - A * x;
p = r;
delta = (r,r);

for( size_t iteration=0UL; iteration<iterations; ++iteration )
{
   Ap = A * p;
   alpha = delta / (p,Ap);
   x += alpha * p;
   r -= alpha * Ap;
   beta = (r,r);
   if( std::sqrt( beta ) < 1E-8 ) break;
   p = r + ( beta / delta ) * p;
   delta = beta;
}
```

PETSc

- Library for large-scale scientific computation
- Large collection of parallel functions for linear solvers, nonlinear solvers, ODE integrators
- Abstracts communication details from user; focus on solving the problem
- Well documented; large collection of online examples/tutorials

PETSc: Parallel Linear Solve Example

```c
#include <petscksp.h>
PETSC_EXTERN PetscErrorCode PCCreate_Jacobi(PC);

int main(int argc,char **args)
{
    Vec               x,b,u;  /* approx solution, RHS, exact solution */
    Mat               A;        /* linear system matrix */
    KSP               ksp;     /* linear solver context */
    PetscReal         norm;     /* norm of solution error */
    PetscInt          i,j,Ii,J,Istart,Iend,m = 8,n = 7,its;
    PetscScalar       v,one = 1.0,neg_one = -1.0;
    PC                pc;      /* preconditioner context */

    PetscInitialize(&argc,&args,(char*)0,help);
    PetscOptionsGetInt(NULL,"-m",&m,NULL);
    PetscOptionsGetInt(NULL,"-n",&n,NULL);

    MatCreate(PETSC_COMM_WORLD,&A);
    MatSetSizes(A,PETSC_DECIDE,PETSC_DECIDE,m*n,m*n);
    MatSetFromOptions(A);
    MatSetUp(A);
```

PETSc: Parallel Linear Solve Example

```
    MatGetOwnershipRange(A,&Istart,&Iend);

    for (Ii=Istart; Ii<Iend; Ii++) {
        v = -1.0; i = Ii/n; j = Ii - i*n;
        if (i>0)   {J = Ii - n; MatSetValues(A,1,&Ii,1,&J,&v,INSERT_VALUES)
        if (i<m-1) {J = Ii + n; MatSetValues(A,1,&Ii,1,&J,&v,INSERT_VALUES)
        if (j>0)   {J = Ii - 1; MatSetValues(A,1,&Ii,1,&J,&v,INSERT_VALUES)
        if (j<n-1) {J = Ii + 1; MatSetValues(A,1,&Ii,1,&J,&v,INSERT_VALUES)
        v = 4.0; MatSetValues(A,1,&Ii,1,&Ii,&v,INSERT_VALUES);
    }

    MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
    MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);

    VecCreate(PETSC_COMM_WORLD,&u);
    VecSetSizes(u,PETSC_DECIDE,m*n);
    VecSetFromOptions(u);
    VecDuplicate(u,&b);
    VecDuplicate(b,&x);
    VecSet(u,one);
    MatMult(A,u,b);
```

PETSc: Parallel Linear Solve Example

```
        KSPCreate(PETSC_COMM_WORLD,&ksp);
        KSPSetOperators(ksp,A,A);
        PCRegister("ourjacobi",PCCreate_Jacobi);
        KSPGetPC(ksp,&pc);
        PCSetType(pc,"ourjacobi");
        KSPSetFromOptions(ksp);

        KSPSolve(ksp,b,x);

        VecAXPY(x,neg_one,u);
        VecNorm(x,NORM_2,&norm);
        KSPGetIterationNumber(ksp,&its);
        PetscPrintf(PETSC_COMM_WORLD,"Norm␣of␣error␣%g␣iterations␣%D\n",
                    (double)norm,its);

        KSPDestroy(&ksp);
        VecDestroy(&u);  VecDestroy(&x);
        VecDestroy(&b);  MatDestroy(&A);
        PetscFinalize();
        return 0;
    }
```

deal.II

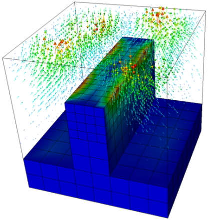


Image Credit: Dr. Wolfgang Bangerth.

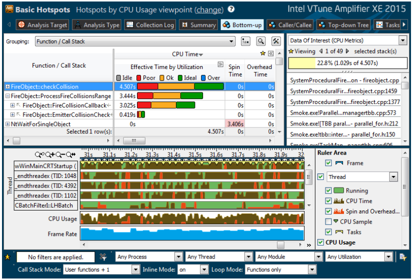
- Modern C++ based library for building applications to solve PDEs with finite elements
- Support for arbitrary degree, adaptive refinement, 1/2/3 spatial dimensions
- Interfaces to a variety of libraries: ARPACK, PETSc, Trilinos, SLEPc, MPI, p4est, METIS, ...
- Well documented code; greater than 50 tutorial programs; online collection of video lectures

Optimization

Optimization

- Getting the most performance out of available hardware; being able to scale efficiently to larger resources
- Libraries are typically optimized; application code can be the bottleneck
- “Premature optimization is the root of all evil” – Donald Knuth; measure code performance and look for critical sections
- Several available tools to provide metrics on performance including CPU, cache utilization, memory bandwidth, communication, ...

Intel VTune



Grouping: Function / Call Stack

Function / Call Stack	CPU Time: Effective Time by Utilization (▨ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over)	Spin Time	Overhead Time
FireObject::checkCollision	4.507s	0s	0s
FireObject::ProcessFireCollisionsRange	3.444s	0s	0s
FireObject::FireCollisionCallback	3.025s	0s	0s
FireObject::EmitterCollisionCheck	0.419s	0s	0s
NtWaitForSingleObject	0s	3.406s	0s
Selected 1 row(s):	4.507s	0s	0s

Data Of Interest (CPU Metrics)

Viewing 1 of 49 selected stack(s)

22.8% (1.029s of 4.507s)

SystemProceduralFire...on - fireobject.cpp
SystemProceduralFir...fireobject.cpp:1459
SystemProceduralFire...fireobject.cpp:1377
Smoke.exe!Parallel...managertbb.cpp:573
Smoke.exe![TBB paral...- parallel_for.h:212
Smoke.exe!tbb::inter...- parallel_for.h:150

Ruler Area
☑ Frame
☑ Thread
☑ Running
☑ CPU Time
☑ Spin and Overhead...
☐ CPU Sample
☑ Tasks
☑ CPU Usage

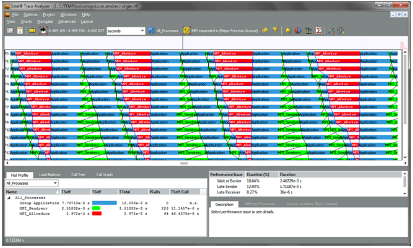
No filters are applied.
Any Process
Any Thread
Any Module
Any Utilization

Call Stack Mode: User functions + 1
Inline Mode: on
Loop Mode: Functions only

Image Credit: Intel.

- Provides CPU metrics, cache misses, thread synchronization information, ...
- Identifies which functions use the most CPU time

Intel Trace Analyzer and Collector



Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
All_Processes					
Group Application	7.74713e-3 s	🟦	13.238e-3 s	0	n.a.
MPI_Sendrecv	2.51892e-3 s	🟩	2.51892e-3 s	226	11.1457e-6 s
MPI_Allreduce	2.972e-3 s	🟥	2.972e-3 s	64	46.4375e-6 s

Performance Issue	Duration (%)	Duration
Wait at Barrier	18.64%	2.46729e-3 s
Late Sender	12.93%	1.71187e-3 s
Late Receiver	0.27%	36e-6 s

Select performance issue to see details.

Image Credit: Intel.

- Collects and reports on MPI communication patterns
- Aids in finding bottlenecks and load balancing issues

Tuning and Analysis Utilities

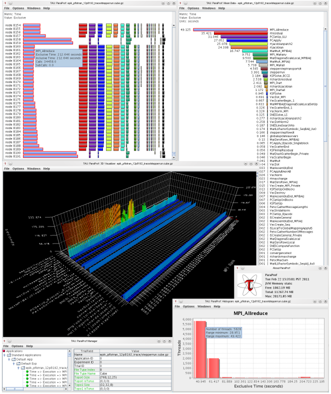


Image Credit: PRACE.

- Open source resource providing similar information as VTune and ITAC, but with a steeper learning curve
- Accesses hardware counters to provide hardware metrics; can instrument MPI calls to trace communication patterns

Additional Resources

Additional Resources

- **Software Carpentry** Lessons on shell, source control, Python, R, SQL
  `http://software-carpentry.org/`
- **HPC Beginner's Guide** More in-depth introduction
  `http://tinyurl.com/korh48z`
- **LLNL Training** Great collection of tutorials and presentations including: MPI, OpenMP, TAU, Python
  `http://tinyurl.com/3zxaw6`
- **deal.II** Video lectures introducing deal.II usage
  `http://www.math.tamu.edu/~bangerth/videos.html`

Additional Resources

- **NICS HPC Seminar** videos, slides, and on campus at Claxton; introduces HPC basics.
  `https://www.nics.tennessee.edu/hpc-seminar-series`
- **MOOCs**
  1. High Performance Scientific Computing – Dr. Randall LeVeque; covers OpenMP, MPI, Python, Fortran. Starts this Friday!
     `https://www.coursera.org/course/scicomp`
  2. Heterogeneous Parallel Computing – Dr. Wen-mei Hwu; covers common parallel algorithm patterns with CUDA
     `https://www.coursera.org/course/hetero`

Contacts at JICS

- Dr. Glenn Brook – Director of AACE and PI of Beacon Project
  glenn-brook@tennessee.edu
- Dr. Lonnie Crosby – Scientific Computing Group Lead and XSEDE ERST
  lcrosby1@utk.edu
- Dr. Ryan Glasby – Computational Engineer and CFD Group Lead
  ryan-glasby@tennessee.edu

Thank You

Shane Sawyer
shane-sawyer@tennessee.edu