

Programming Models for High Performance Scientific Computing

Jeff Hammond

Leadership Computing Facility
Argonne National Laboratory

8 June 2010



Disclaimer

I wrote the published abstract when scheduled for July. This talk covers more introductory material and less implementation detail. Please interrupt frequently.

These slides are shamefully texty for posterity. If you want pictures, I will use the whiteboard.

All opinions are my own.

My background and other details

I was trained as a chemist but got pretty deep into computer science through my work on NWChem, which utilizes a variety of non-trivial software technologies (e.g. automatic code generation, GAS programming model, one-sided communication).

My slides and contact info are posted here:
<https://wiki.alcf.anl.gov/index.php/User:Jhammond>

Parallelism as a concept

What is parallelism?

Parallelism in computing generally refers to doing multiple things at the same time.

Parallelism necessarily involves hardware.

Algorithms are not parallel, rather concurrent, but everyone uses the terms interchangeably.

Concurrency in computing is not defined unambiguously; asynchronity (uncorrelated in time) is more useful.

Getting dressed in parallel I

How many operations does it take to put on a suit?

Tasks: shirt, jacket, pants, socks, shoes, tie, belt, fedora.

In serial, one needs 8 steps to put on this outfit.

Dependences: shirt \rightarrow tie, tie \rightarrow jacket, socks \rightarrow shoes, pants \rightarrow belt, pants \rightarrow shoes, pants \leftrightarrow socks.

If we analyze concurrency, what is the best we can do in parallel?

Audience participation time

Getting dressed in parallel II

Thread	Step 1	Step 2	Step 3
1	shirt	tie	jacket
2	pants	belt	fedora
3	-	socks	shoes

This solution is not unique but it is optimal.

Thread	Step 1	Step 2	Step 3
1	shirt	tie	jacket
2	-	pants	belt
3	socks	fedora	shoes

Because three steps are sequential, one cannot do better than that overall.

Getting dressed in parallel III

But if you want to maximize efficiency of your overall usage:

Thread	Step 1	Step 2	Step 3	Step 4
1	shirt	tie	jacket	fedora
2	socks	pants	belt	shoes

With an increasing emphasis on power efficiency, one may trade parallelism for efficiency at the chip level. One has to decide if the 12.5% overhead of the ideal stage in the maximally-parallel execution is worth the 25% decrease in execution time. It looks stupid here, but what if 1M people are getting dressed in parallel?

Barriers to parallelism

Perfect parallelism I

There is no such thing as perfect parallelism unless you remove yourself from the equation. Fully asynchronous execution leaves you as the bottleneck.

The most (computationally) scalable form of parallelism involves an arbitrarily large number of uncoupled tasks.

Perfect parallelism II

Example: input a list chemical structures, compute the “score” of this compound with respect to binding with a given protein (the drug-design problem).

This works in part because size of the input and output data are small with respect to the intermediate computation.

Data dependencies

As in the suit example, parallelism is bounded by concurrency, i.e. the maximum number of asynchronous execution paths the algorithm has.

Enforcing data-dependencies with fine-granularity is expensive due to bookkeeping overhead.

Enforcing data-dependencies with coarse-granularity is expensive due to global sync e.g. `MPI_Barrier()`.

Communication overhead

Except for nearest-neighbor-exchange, communication does not scale. All networks drag under heavy-load (contention).

If everyone talks to one node, that node will be overwhelmed as “everyone” becomes larger.

Collectives usually don't scale. Even if you have a tree network ala Blue Gene, memory becomes an issue.

Load-balancing issues

It does not matter which process finishes first — the job is done when the *last* process finishes.

The same problem applies to every single synchronous operation in your code, including collectives and blocking `send/recv`.

Most of the problem with collectives is not the explicit time due to communication but implicit time due to waiting on last process.

$$T_{total} = T_{serial} + \frac{T_{parallel}}{N_{proc}}$$

Strong-scaling (Amdahl):

$$\lim_{N \rightarrow \infty} T_{total} \approx T_{serial}$$

Weak-scaling (Gustafson):

$$\lim_{N \rightarrow \infty} T_{total} \approx \frac{T_{parallel}(N_{proc})}{N_{proc}}$$

The programming model that isn't

MPI is not a programming model. Just because smart people say it is does not make it true.

MPI is a communication protocol and does not care what your data or execution model is.

Every programming model implementation that exists today has been implemented using MPI, including Charm++, Global Arrays and PGAS (UPC, Chapel, X10).

Basic MPI usually the simple model of local-data/local-execution with message-passing.

Master-worker programming model using collectives or spinning-master is easy.

You can make MPI do anything with threads.

Parallelism in practice

Examples of parallel applications (paradigms)

_____ (Swift):

<http://www.ci.uchicago.edu/swift/>

GFMC (ADLB):

<http://www.cs.mtsu.edu/~rbutler/adlb/>

GPAW (MPI/ScaLAPACK):

<https://wiki.fysik.dtu.dk/gpaw/>

NWChem (GA/ARMCI):

<http://www.emsl.pnl.gov/capabilities/computing/nwchem/>

<http://www.emsl.pnl.gov/docs/global/>

<http://www.emsl.pnl.gov/docs/parsoft/armci/>

NAMD (Charm++):

<http://www.ks.uiuc.edu/Research/namd/>

<http://charm.cs.uiuc.edu/>

What is Swift?

Swift is a language which expresses a specific type of parallelism in a very natural way:

- Mapping of filesystem concepts within the language
- Explicit expression of data-dependencies
- Run-time automation of parallel execution of independent tasks
- Run-time manages filesystem access in a smart way
- Does not attempt to support features for other paradigms (e.g. message-passing)

Why use Swift?

- It is not trivial to implement this type asynchronous parallelism in MPI
- Master-worker does not scale past a few thousand processes (see GFMC)
- If you write a script to run 100K file-accessing tasks on your own, you will kill the filesystem
- If you need a thumbtack, use a thumbtack, not a nailgun

What is GFMC?

GFMC is the officially unnamed Green's Function Monte Carlo code of Steve Pieper et al. for modeling of nucleon-nucleon interactions in atomic nuclei.

- Monte Carlo integration is loosely-coupled
- GFMC is defined by a task pool and some small global state for convergence
- Originally used a master-worker but the worker becomes overwhelmed with task-delegation
- ADLB implements a distributed shared task-queue to ameliorate the communication associated with delegation

What is GPAW?

GPAW is a real-space density-function theory (DFT) code for material science applications.

GPAW solves the Kohn-Sham equations iteratively, which involves a whole lot of dense linear algebra (matrix-matrix multiplication aka GEMM, inverse Cholesky and diagonalization aka DIAG).

Dense linear algebra can be easy (GEMM) or hard (DIAG). GEMM is easy because it is data-parallel whereas DIAG is highly synchronous.

Anatomy of GEMM

Matrix-matrix multiplication is effectively data-parallel:

```
forall i,j,k  
    C(i,j) += A(i,k) * B(k,j)
```

Scalar multiplication leads to terrible performance, so do a bunch of local computation at once with BLAS:

```
forall I,J,K (tiles)  
    C(I,J) += A(I,K) * B(K,j) ← DGEMM
```

If the tiles are distributed on a process-grid, scaling is straightforward.

What is NWChem?

NWChem implements a myriad of computational chemistry methods using Global Arrays.

Assumption: Chemists \neq (good) programmers.

Solution 1: Make a parallel computer behave like a serial computer. More on this later.

Solution 2: Have non-chemists implement non-chemistry functionality as a black-box.

Solution 3: Implement all core objects with C++-style abstraction but in F77.

Disguising a parallel computer

What programming complex algorithms in parallel difficult?

- Bookkeeping distributed data.
Ex: what process owns A(235)?
- Managing two-sided communication.
Ex: you cannot request (receive) data without knowing on the remote site to do a send — not like a filesystem.
- Irregular algorithms require dynamic load-balancing. Can we do this without master-worker?

The Global Arrays (GA) programming model

Arrays are distributed across machine automatically
— can be tiled arbitrarily for different uses.

Data is access via rank-free Put and Get in analogy
to read and write for files. Host process not
required to send (recv) data for Get (Put).

Many numerical computations require $c+=a*b$ so
GA implement accumulate (Acc) as well. Shared
counters implemented using read-modify-write
(Rmw).

Provides parallel dense linear algebra via PeIGS,
ScaLAPACK or SRUMMA.

Using Global Arrays

```
<GA(double)> A,B,C, <buf(double)> a,b,c  
<GA(int)> N=0, buf(int)> n  
forall chunks/tiles of A, B, C:  
  Rmw(n←N++)  
  if test(n):  
    Get(a←A)  
    Get(b←B)  
    c = foo(a,b)  
    Acc(c→C)
```

Why use Global Arrays?

- Provides trivial way to implement global state.
- Memory-bound and want to avoid I/O.
- Unpredictable communication patterns are easy with one-sided.
- Avoid bookkeeping global data entirely.

What is NAMD?

NAMD is a really popular molecular dynamics (MD) code. Physically-meaningful MD requires 10^{6-9} timesteps because atomic timescale is femtoseconds.

There are 86,400 seconds in a day. If you want 10^6 timesteps, that is 11.5 days. If one wants millisecond MD, a single timestep must take approximately 1 microsecond.

Challenges for NAMD

MD becomes load-imbalanced quickly as atoms move. Who owns the atom if it moves from one domain to another?

Global communication is required to do long-range Coulomb interactions.

Remember that load-imbalance + global communication = very bad.

What is Charm++

Charm++ is the parallel run-time system co-designed with NAMD.

Unlike MPI or GA, Charm++ has a message-driven execution model.

Message-driven execution makes sense if communication is more important than computation.

Charm++ load-balances iteratively by moving tasks around as needed.

What is Charm++

Charm++ is the parallel run-time system co-designed with NAMD.

Unlike MPI or GA, Charm++ has a message-driven execution model.

Message-driven execution makes sense if communication is more important than computation.

Charm++ load-balances iteratively by moving tasks around as needed.

Conclusions

Factors which affect parallelism

How regular is your computation?

How regular is your communication?

What are your communication patterns?

Do you actually understand what your algorithm is doing?

Commence tomato throwing!