Parallel Programming Methodologies

John Urbanic
Parallel Computing Scientist
Pittsburgh Supercomputing Center
Why this talk?

This is a version of a presentation I give at the beginning of the International HPC Summer School track.

And I give at the end of my Summer Boot Camp to help people understand which of the techniques they have learned best apply to their own applications.

Warning

This version contains blunt editorializing.
FLOPS we need: Climate change analysis

- Cloud resolution, quantifying uncertainty, understanding tipping points, etc., will drive climate to exascale platforms
- New math, models, and systems support will be needed

- “Reanalysis” projects need 100× more computing to analyze observations
- Machine learning and other analytics are needed today for petabyte data sets
- Combined simulation/observation will empower policy makers and scientists

Courtesy Horst Simon, LBNL
## ECP application domains.

<table>
<thead>
<tr>
<th>National security</th>
<th>Energy security</th>
<th>Economic security</th>
<th>Scientific discovery</th>
<th>Earth system</th>
<th>Health care</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stockpile stewardship</td>
<td>Turbine wind plant efficiency</td>
<td>Additive manufacturing of</td>
<td>Find, predict, and control</td>
<td>Accurate regional impact assessments in Earth</td>
<td>Accelerate and translate cancer research</td>
</tr>
<tr>
<td>Next generation simulation tools</td>
<td>High-efficiency, low-emission</td>
<td>qualified metal parts</td>
<td>materials and properties</td>
<td>system models</td>
<td></td>
</tr>
<tr>
<td>for assessing nuclear</td>
<td>combustion engine and gas turbine</td>
<td>Reliable and efficient</td>
<td>Cosmological probe of the</td>
<td>Stress-resistant crop analysis and catalytic</td>
<td></td>
</tr>
<tr>
<td>weapons performance</td>
<td>design</td>
<td>planning of the power</td>
<td>standard model of particile</td>
<td>conversion of biomass-derived alcohols</td>
<td></td>
</tr>
<tr>
<td>Response to hostile threat</td>
<td>Materials design for extreme</td>
<td>Seismic hazard risk</td>
<td>Validate fundamental</td>
<td>Metagenomics for analysis of biogeochemical</td>
<td></td>
</tr>
<tr>
<td>environments and reentry conditions</td>
<td>environments of nuclear fission and</td>
<td>assessment</td>
<td>laws of nature</td>
<td>cycles, climate change, environmental</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fusion reactors</td>
<td></td>
<td>Demystify origin of chemical</td>
<td>remediation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Design and commercialization of Small</td>
<td></td>
<td>elements</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Modular Reactors</td>
<td></td>
<td>Light source-enabled analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Subsurface use for carbon capture,</td>
<td></td>
<td>of protein and molecular</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>petroleum extraction, waste disposal</td>
<td></td>
<td>structure and design</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Scale-up of clean fossil fuel</td>
<td></td>
<td>Whole-device model of</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>combustion</td>
<td></td>
<td>magnetically confined fusion</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Biofuel catalyst design</td>
<td></td>
<td>plasmas</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Moore's Law abandoned serial programming around 2004
Come back tomorrow....

Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp
<table>
<thead>
<tr>
<th>Processor</th>
<th>Year</th>
<th>Vector</th>
<th>Bits</th>
<th>SP FLOPs / core / cycle</th>
<th>Cores</th>
<th>FLOPs/cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentium III</td>
<td>1999</td>
<td>SSE</td>
<td>128</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Pentium IV</td>
<td>2001</td>
<td>SSE2</td>
<td>128</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Core</td>
<td>2006</td>
<td>SSE3</td>
<td>128</td>
<td>8</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>Nehalem</td>
<td>2008</td>
<td>SSE4</td>
<td>128</td>
<td>8</td>
<td>10</td>
<td>80</td>
</tr>
<tr>
<td>Sandybridge</td>
<td>2011</td>
<td>AVX</td>
<td>256</td>
<td>16</td>
<td>12</td>
<td>192</td>
</tr>
<tr>
<td>Haswell</td>
<td>2013</td>
<td>AVX2</td>
<td>256</td>
<td>32</td>
<td>18</td>
<td>576</td>
</tr>
<tr>
<td>KNC</td>
<td>2012</td>
<td>AVX512</td>
<td>512</td>
<td>32</td>
<td>64</td>
<td>2048</td>
</tr>
<tr>
<td>KNL</td>
<td>2016</td>
<td>AVX512</td>
<td>512</td>
<td>64</td>
<td>72</td>
<td>4608</td>
</tr>
<tr>
<td>Skylake</td>
<td>2017</td>
<td>AVX512</td>
<td>512</td>
<td>96</td>
<td>28</td>
<td>2688</td>
</tr>
</tbody>
</table>
MPPs (Massively Parallel Processors)

Distributed memory at largest scale. Shared memory at lower level.

**Summit (ORNL)**
- 122 PFlops Rmax and 187 PFlops Rpeak
- IBM Power 9, 22 core, 3GHz CPUs
- 2,282,544 cores
- NVIDIA Volta GPUs
- EDR Infiniband

**Sunway TaihuLight (NSC, China)**
- 93 PFlops Rmax and 125 PFlops Rpeak
- Sunway SW26010 260 core, 1.45GHz CPU
- 10,649,600 cores
- Sunway interconnect
Many Levels and Types of Parallelism

- Vector (SIMD)
- Instruction Level (ILP)
  - Instruction pipelining
  - Superscaler (multiple instruction units)
  - Out-of-order
  - Register renaming
  - Speculative execution
  - Branch prediction

- Multi-Core (Threads)
- SMP/Multi-socket
- Accelerators: GPU & MIC
- Clusters
- MPPs

Also Important
- ASIC/FPGA/DSP
- RAID/IO

OpenMP 4/5 can help!

Compiler
(not your problem)
Parallel Computing

One woman can make a baby in 9 months.

Can 9 women make a baby in 1 month?

But 9 women can make 9 babies in 9 months.

First two bullets are Brook’s Law. From *The Mythical Man-Month.*
Need to write some scalable code?

First Choice:

Pick a language - or maybe a library, or paradigm (whatever that is)?
Languages: Pick One *(Hint: MPI + ?)*

Parallel Programming environments since the 90's

**Alternative Approach**

“When all you have is a hammer, everything looks like a nail.”
Prototypical Application: Serial Weather Model
First parallel Weather Modeling algorithm: Richardson in 1917

Courtesy John Burkhardt, Virginia Tech
Four meteorologists in the same room sharing the map.

Fortran:

```fortran
!$omp parallel do
do i = 1, n
    a(i) = b(i) + c(i)
enddo
```

C/C++:

```c
#pragma omp parallel for
for(i=1; i<=n; i++)
    a[i] = b[i] + c[i];
```
1 meteorologists coordinating 1000 math savants using tin cans and a string.
call MPI_Send( numbertosend, 1, MPI_INTEGER, index, 10, MPI_COMM_WORLD, errcode)

- 
- 

call MPI_Recv( numbertoreceive, 1, MPI_INTEGER, 0, 10, MPI_COMM_WORLD, status, errcode)

- 
- 

call MPI_Barrier(MPI_COMM_WORLD, errcode)

50 meteorologists using a telegraph.
MPI as an answer to an emerging problem ?!

We are at a historic crossover where the cost of even on-chip data movement is surpassing the cost of computation.

MPI codes explicitly acknowledge and manage data locality and movement (communication).

Both this paradigm, and quite possible outright MPI, offer the only immediate solution. You and your programs may find a comfortable future in the world of massive multi-core.

This is a somewhat recent realization in the most avant-garde HPC circles. Amaze your friends with your insightful observations!
The pieces fit like this…

OpenMP

OpenACC

MPI
<table>
<thead>
<tr>
<th>#</th>
<th>Site</th>
<th>Manufacturer</th>
<th>Computer</th>
<th>CPU Interconnect [Accelerator]</th>
<th>Cores</th>
<th>Rmax (Tflops)</th>
<th>Rpeak (Tflops)</th>
<th>Power (MW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RIKEN Center for Computational Science</td>
<td>Fujitsu</td>
<td>Fugaku</td>
<td>ARM 8.2A+ 48C 2.2GHz Torus Fusion Interconnect</td>
<td>7,299,072</td>
<td>415,530</td>
<td>513,854</td>
<td>28.3</td>
</tr>
<tr>
<td></td>
<td>Japan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>DOE/SC/ORNL</td>
<td>IBM</td>
<td>Summit</td>
<td>Power9 22C 3.0 GHz Dual-rail Infiniband EDR NVIDIA V100</td>
<td>2,414,592</td>
<td>148,600</td>
<td>200,794</td>
<td>10.1</td>
</tr>
<tr>
<td></td>
<td>United States</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>DOE/NNSA/LLNL</td>
<td>IBM</td>
<td>Sierra</td>
<td>Power9 3.1 GHz 22C Infiniband EDR NVIDIA V100</td>
<td>1,572,480</td>
<td>94,640</td>
<td>125,712</td>
<td>7.4</td>
</tr>
<tr>
<td></td>
<td>United States</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>National Super Computer Center in Wuxi</td>
<td>NRCPC</td>
<td>Sunway TaihuLight</td>
<td>Sunway SW26010 260C 1.45GHz</td>
<td>10,649,600</td>
<td>93,014</td>
<td>125,435</td>
<td>15.3</td>
</tr>
<tr>
<td></td>
<td>China</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>National Super Computer Center in Guangzhou</td>
<td>NUDT</td>
<td>Tianhe-2</td>
<td>Intel Xeon ES-2692 2.2 GHz TH Express-2 Intel Xeon Phi 31S1P</td>
<td>4,981,760</td>
<td>61,444</td>
<td>100,678</td>
<td>18.4</td>
</tr>
<tr>
<td></td>
<td>China</td>
<td></td>
<td>(MilkyWay-2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Eni S.p.A</td>
<td>Dell</td>
<td>HPc5</td>
<td>Xeon 24C 2.1 GHz Infiniband HDR NVIDIA V100</td>
<td>669,760</td>
<td>35,450</td>
<td>51,720</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>Italy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Eni S.p.A</td>
<td>NVIDIA</td>
<td>Selene</td>
<td>EPYC 64C 2.25GHz Infiniband HDR NVIDIA A100</td>
<td>272,800</td>
<td>27,580</td>
<td>34,568</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td>Italy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Texas Advanced Computing Center/Univ. of</td>
<td>Dell</td>
<td>Frontera</td>
<td>Intel Xeon 8280 28C 2.7 GHz InfiniBand HDR</td>
<td>448,448</td>
<td>23,516</td>
<td>38,745</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Texas</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Cineca</td>
<td>IBM</td>
<td>Marconi100</td>
<td>Power9 16C 3.0 GHz Infiniband EDR NVIDIA V100</td>
<td>347,776</td>
<td>21,640</td>
<td>29,354</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td>Italy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Swiss National Supercomputing Centre (CSCS)</td>
<td>Cray</td>
<td>Piz Daint Cray</td>
<td>Xeon E5-2690 2.6 GHz Aries NVIDIA P100</td>
<td>387,872</td>
<td>21,230</td>
<td>27,154</td>
<td>2.4</td>
</tr>
<tr>
<td></td>
<td>Switzerland</td>
<td></td>
<td>XC50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Other Paradigms?

- Message Passing
  - MPI
- Threads
  - OpenMP, OpenACC, CUDA
- Hybrid
  - MPI + OpenMP
- Data Parallel
  - Fortran90
- PGAS (Partitioned Global Address Space)
  - UPC, Coarray Fortran (Fortran 2008)
- Frameworks
  - Charm++
Data Parallel – Fortran90

Computation in FORTRAN 90

```
Real A(4,4), B(4,4), C(4,4)
A=2.0
FORALL (I=1:4, J=1:4)
  B(I,J)=I+J
C=A+B
```

P = Processor

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
</tbody>
</table>

\[ C = A + B \]

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
Data Parallel Communication in FORTRAN 90

```
P P P P P
P P P P P
P P P P P
P P P P P
P = Processor

Real A(4,4), B(4,4)
FORALL (I=1:4, J=1:4)
  B(I, J) = I + J
A = CSHIFT (B, DIM=2, 1)

A =

3 4 5 6
4 5 6 7
5 6 7 8
2 3 4 5

CSHIFT (B, 2, 1)

2 3 4 5
3 4 5 6
4 5 6 7
5 6 7 8
```
Data Parallel

Pros

- So simple you just learned some of it
- ...or already knew it from using Fortran
- Easy to debug

Cons

- If your code doesn’t totally, completely express itself as nice array operations, you are left without a flexible alternative.
  - Image processing: Great
  - Irregular meshes: Not so great
Domain Decomposition Done Well: Load Balanced

- A parallel algorithm can only be as fast as the slowest chunk.
  - Might be dynamic (hurricane moving up coast)
- Communication will take time
  - Usually orders of magnitude difference between registers, cache, memory, network/remote memory, disk
  - Data locality and “neighborly-ness” matters very much.

Is Texas vs. New Jersey a good idea?
PGAS with Co-Array Fortran (now Fortran 2008)

Co-array synchronization is at the heart of the typical Co-Array Fortran program. Here is how to exchange an array with your north and south neighbors:

```fortran
COMMON/XCTILB4/ B(N,4)[*]
SAVE /XCTILB4/

CALL SYNC_ALL( WAIT=(/IMG_S,IMG_N/) )
B(:,3) = B(:,1)[IMG_S]
B(:,4) = B(:,2)[IMG_N]
CALL SYNC_ALL( WAIT=(/IMG_S,IMG_N/) )
```

Lots more examples at co-array.org.
Partitioned Global Address Space: (PGAS)

Multiple threads share at least a part of a global address space.

Can access local and remote data with same mechanisms.

Can distinguish between local and remote data with some sort of typing.

Variants:
- Co-Array Fortran (CAF)
- Fortran 2008
- Unified Parallel C (UPC)

Pros:
1. Looks like SMP on a distributed memory machine.
2. Currently translates code into an underlying message passing version for efficiency.

Cons:
1. Depends on (2) to be efficient.
2. Can easily write lots of expensive remote memory access without paying attention.
3. Currently immature.
Frameworks

One of the more experimental approaches that was gaining some traction was to use a parallel framework that handle the load balancing and messaging while you “fill in” the science. Charm++ is the most popular example:

**Charm++**

- Object-oriented parallel extension to C++
- Run-time engine allows work to be “scheduled” on the computer.
- Highly-dynamic, extreme load-balancing capabilities.
- Completely asynchronous.
- NAMD, a very popular MD simulation engine is written in Charm++

![CHARM++: A high level view](image)
Frameworks (Newsflash!)

• After a long time with no positive reports in this space, I can definitely say that the Machine Learning (Artificial Intelligence) community has embraced this in an effective manner.

• The most popular frameworks/toolkits/packages used for deep learning (aka Neural Nets) are very much in this philosophy.

• Caffe, TensorFlow and others use a high level descriptive approach to arrange other components, often themselves a higher level layer in Python or whatnot, to invoke libraries written in C++ (and actually Fortran is hidden in there more often than those groups would believe in the form of BLAS type libraries).

• These frameworks use threads, GPUs and distributed nodes very heavily.

• You could say that the math library nature of this work makes this unique, but the innovation in arranging these codeflows is not at all rote.
Some Alternatives

- **OpenCL (Khronos Group)**
  - Everyone supports, but not as a primary focus
  - Intel – OpenMP
  - NVIDIA – CUDA, OpenACC
  - AMD – now HIP
  - Khronos has now brought out SYCL

- Fortran 2008+ threads (sophisticated but not consistently implemented)

- C++11 threads are basic (no loops) but better than POSIX
  - C++17 brings parallel STL
  - C++20 atomic smart pointers, futures, latches and barriers, coroutines, transactional memory, task blocks

- Python threads are fake (due to Global Interpreter Lock) but multiprocessing is pretty easy.

- DirectCompute (Microsoft) is not HPC oriented

- C++ AMP (MS/AMD)

- TBB (Intel C++ template library)

- Cilk (Intel, now in a gcc branch)

- Intel oneAPI (Includes DPC++ and extends SYCL)

- Kokkos
How parallel is a code?

- Parallel performance is defined in terms of scalability

**Strong Scalability**
Can we get faster for a given problem size?

**Weak Scalability**
Can we maintain runtime as we scale up the problem?
Amdahl’s Law

- If there is x% of serial component, speedup cannot be better than 100/x.
- If you decompose a problem into many parts, then the parallel time cannot be less than the largest of the parts.
- If the critical path through a computation is T, you cannot complete in less time than T, no matter how many processors you use.

- Amdahl's law used to be cited by the knowledgeable as a limitation.
- These days it is mostly raised by the uninformed.
- Massive scaling is commonplace:
  - Science Literature
  - Web (map reduce everywhere)
  - Data Centers (Spark, etc.)
  - Machine Learning (GPUs and others)
Weak vs. Strong scaling

More Processors

Weak Scaling

More accurate results

Strong Scaling

More Processors

Faster results
(Tornado on way!)
End of Moore’s Law Will Lead to New Architectures

Non-von Neumann Architectures

Neuromorphic Computing

Quantum Computing

Beyond CMOS Technology

TODAY

CMOS
The Future and where you fit.

While the need is great, there is only a short list of serious contenders for 2020 exascale computing usability.

**MPI 3.0 +X** (MPI 3.0 specifically addresses exascale computing issues)

**PGAS** (partitioned global address space)

**CAF** (now in Fortran 2008!)

**UPC**

What about Big Data?

Deep Learning?

A Different Talk!

Perhaps next year?
Again...

Hybrid computing is important!