Outro To Parallel Computing

John Urbanic
Parallel Computing Scientist
Pittsburgh Supercomputing Center
Purpose of this talk

Now that you know how to do some real parallel programming, you may wonder how much you don’t know. With your newly informed perspective we can now take a meaningful look at the parallel software landscape so that you can see how much of it you are equipped to traverse.
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?
Weak vs. Strong scaling

Weak Scaling:
- More Processors
- More accurate results

Strong Scaling:
- More Processors
- Faster results (Tornado on way!)
How many processors can we really use?

Let’s say we have a legacy code such that is it only feasible to convert half of the heavily used routines to parallel:
Amdahl’s Law

If we run this on a parallel machine with five processors:

Our code now takes about 60s. We have sped it up by about 40%.

Let’s say we use a thousand processors:

We have now sped our code by about a factor of two. Is this a big enough win?
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)
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

<table>
<thead>
<tr>
<th>ABCPL</th>
<th>CORRELATE</th>
<th>GLU</th>
<th>Montat</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACE</td>
<td>CPS</td>
<td>GUARD</td>
<td>Legion</td>
</tr>
<tr>
<td>ACT++</td>
<td>CRL</td>
<td>HAdL</td>
<td>Meta Chaos</td>
</tr>
<tr>
<td>Active messages</td>
<td>CSP</td>
<td>Haskell</td>
<td>Midway</td>
</tr>
<tr>
<td>Adl</td>
<td>Cthreads</td>
<td>HPC++</td>
<td>Multipede</td>
</tr>
<tr>
<td>Adomith</td>
<td>CUMULUS</td>
<td>JAVAR</td>
<td>CupPar</td>
</tr>
<tr>
<td>ADDAP</td>
<td>DAGGER</td>
<td>HORUS</td>
<td>Mirage</td>
</tr>
<tr>
<td>AFAPI</td>
<td>DAPPLE</td>
<td>HPC</td>
<td>MpC</td>
</tr>
<tr>
<td>ALWAN</td>
<td>Data Parallel C</td>
<td>IMPACT</td>
<td>MORSIX</td>
</tr>
<tr>
<td>AM</td>
<td>DC++</td>
<td>INST</td>
<td>Modula-P</td>
</tr>
<tr>
<td>AMDC</td>
<td>DCE++</td>
<td>JAVAR</td>
<td>Multipol</td>
</tr>
<tr>
<td>AppLeS</td>
<td>DDDD</td>
<td>JavaKMI</td>
<td>MPI</td>
</tr>
<tr>
<td>Amdoba</td>
<td>DICE</td>
<td>javaPG</td>
<td>MPC++</td>
</tr>
<tr>
<td>ARTS</td>
<td>DIPC</td>
<td>JavaSpace</td>
<td>Mumin</td>
</tr>
<tr>
<td>Athapascan-0b</td>
<td>DOLIB</td>
<td>JIDL</td>
<td>Nano-Threads</td>
</tr>
<tr>
<td>Aurora</td>
<td>DOME</td>
<td>Joyce</td>
<td>NECL</td>
</tr>
<tr>
<td>Autosmp</td>
<td>DOSMOS</td>
<td>Khoros</td>
<td>NetClasses++</td>
</tr>
<tr>
<td>Blaze</td>
<td>DRL</td>
<td>KORMA</td>
<td>Nexus</td>
</tr>
<tr>
<td>BSP</td>
<td>DSM-Threads</td>
<td>KOAN/Fortran-S</td>
<td>Ninfrod</td>
</tr>
<tr>
<td>BlockComm</td>
<td>Ease</td>
<td>LAM</td>
<td>NOW</td>
</tr>
<tr>
<td>C-</td>
<td>ECO</td>
<td>Linde</td>
<td>Objective Linda</td>
</tr>
<tr>
<td>&quot;C++&quot;</td>
<td>Eiffel</td>
<td>JADA</td>
<td>Occam</td>
</tr>
<tr>
<td>C++</td>
<td>Eilean</td>
<td>WWWinden</td>
<td>Omega</td>
</tr>
<tr>
<td>Cashmere</td>
<td>Emerald</td>
<td>ISETL-Linda</td>
<td>OpenMP</td>
</tr>
<tr>
<td>C4</td>
<td>EPL</td>
<td>ParLin</td>
<td>Orcia</td>
</tr>
<tr>
<td>CC++</td>
<td>Excalibur</td>
<td>Eilean</td>
<td>OOP90</td>
</tr>
<tr>
<td>Chn</td>
<td>Express</td>
<td>FPL</td>
<td>P+</td>
</tr>
<tr>
<td>Charlotte</td>
<td>Falcon</td>
<td>Fused-Lnda</td>
<td>P4-Lnda</td>
</tr>
<tr>
<td>Charn</td>
<td>Filaments</td>
<td>Flash</td>
<td>Pablo</td>
</tr>
<tr>
<td>Charn++</td>
<td>FM</td>
<td>The FORCE</td>
<td>PADE</td>
</tr>
<tr>
<td>Cil</td>
<td>Flash</td>
<td>Fork</td>
<td>PAIDE</td>
</tr>
<tr>
<td>Cilk</td>
<td>Forktran-M</td>
<td>Fx</td>
<td>Papers</td>
</tr>
<tr>
<td>CM-Fortran</td>
<td>GA</td>
<td>GA</td>
<td>AFAPI</td>
</tr>
<tr>
<td>Converse</td>
<td>GAMMA</td>
<td>Glenda</td>
<td>Pansi++</td>
</tr>
<tr>
<td>Code</td>
<td>Gamma</td>
<td>Glenda</td>
<td>Paradigm</td>
</tr>
<tr>
<td>COOL</td>
<td>Glenda</td>
<td>GLU</td>
<td>Parafrase2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GUID</td>
<td>Parallel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HAdL</td>
<td>Parallel++</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Haskell</td>
<td>ParC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HPC++</td>
<td>ParLib++</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JAVAR</td>
<td>ParLin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HORUS</td>
<td>Parmacs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HPC</td>
<td>Pari</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IMPACT</td>
<td>pC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INST</td>
<td>pC++</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JavaKMI</td>
<td>PCN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>javaPG</td>
<td>PCE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JavaSpace</td>
<td>PEACE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JIDL</td>
<td>PCU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Joyce</td>
<td>PET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Khoros</td>
<td>PETSx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KORMA</td>
<td>PENNY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Linde</td>
<td>Phosphorus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JADA</td>
<td>POET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WWWinden</td>
<td>Polaris</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISETL-Linda</td>
<td>POOMA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ParLin</td>
<td>POOL-T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eilean</td>
<td>PRESTO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FPL</td>
<td>P-RIO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Glenda</td>
<td>Prospero</td>
</tr>
<tr>
<td></td>
<td></td>
<td>POSYBL</td>
<td>Proteus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Objective-Linda</td>
<td>QPC++</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LIPS</td>
<td>PVM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Locust</td>
<td>PSI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lpar</td>
<td>PSIDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lpar</td>
<td>Quake</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Local</td>
<td>Quick</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maxine</td>
<td>QuickThreads</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maxtold</td>
<td>Sage++</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parallelogram</td>
<td>SCANDAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Montat</td>
<td>SAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Legion</td>
<td>pC++</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meta Chaos</td>
<td>SCHEDULE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Midway</td>
<td>SciTiL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Multipede</td>
<td>POET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CupPar</td>
<td>SDDA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mirage</td>
<td>SHMEM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MpC</td>
<td>SIMPLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Morsix</td>
<td>Sma</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modula-P</td>
<td>SISAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Multipol</td>
<td>distributed smalltalk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPI</td>
<td>SMIL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPC++</td>
<td>SONIC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mumin</td>
<td>Split-C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nano-Threads</td>
<td>SR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NECL</td>
<td>Shthreads</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NetClasses++</td>
<td>Strand</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nexus</td>
<td>SUIF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ninfrod</td>
<td>Synergy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOW</td>
<td>Telegrphos</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Objective Linda</td>
<td>TCGMSG.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Occam</td>
<td>Threads++.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Omega</td>
<td>TreadMarks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OpenMP</td>
<td>TRAPPER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Orcia</td>
<td>uC++</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OOP90</td>
<td>UNITY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P+</td>
<td>UC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P4-Lnda</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>p4-Lnda</td>
<td>VCC++</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pablo</td>
<td>Vedi1D V-NUS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PADE</td>
<td>VPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAIDE</td>
<td>Win32 threads</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Papers</td>
<td>WinPar</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Papers</td>
<td>WWWindia</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AFAPI</td>
<td>XENOOPS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pansi++</td>
<td>XPC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Paradigm</td>
<td>Zounds</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Montat</td>
<td>ZPL</td>
</tr>
</tbody>
</table>
Paradigm?

- Message Passing
  - MPI
- Data Parallel
  - Fortran90
- Threads
  - OpenMP, OpenACC, CUDA, OpenCL, SYCL
- PGAS
  - UPC, Coarray Fortran
- Frameworks
  - Charm++
- Hybrid
  - MPI + OpenMP
Message Passing: MPI in particular

Pros

- Has been around a longtime (~20 years inc. PVM)
- Dominant
- Will be around a longtime (on all new platforms/roadmaps)
- Lots of libraries
- Lots of algorithms
- Very scalable (100K+ cores right now)
- Portable
- Works with hybrid models
- We teach MPI in two days also
- This is the only route to massive scalability today!

Cons

- Lower level means more detail for the coder
- Debugging requires more attention to detail
- Domain decomposition and memory management must be explicit
- Students leaving our MPI workshop may face months of work before they are able to actually run their production code
- Development usually requires a “start from scratch” approach
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

\[
\begin{array}{cccc}
4 & 5 & 6 & 7 \\
5 & 6 & 7 & 8 \\
6 & 7 & 8 & 9 \\
7 & 8 & 9 & 10
\end{array}
\quad
\begin{array}{cccc}
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2
\end{array}
\quad
\begin{array}{cccc}
2 & 3 & 4 & 5 \\
3 & 4 & 5 & 6 \\
4 & 5 & 6 & 7 \\
5 & 6 & 7 & 8
\end{array}
\]
Data Parallel
Communication in FORTRAN 90

\[ \begin{align*}
\text{Real } A(4,4), B(4,4) \\
\text{FORALL } (I=1:4, J=1:4) \\
B(I, J) &= I + J \\
A &= \text{CSHIFT } (B, \text{ DIM}=2, 1) \\
P &= \text{Processor}
\end{align*} \]

\[
\begin{array}{cccc}
3 & 4 & 5 & 6 \\
4 & 5 & 6 & 7 \\
5 & 6 & 7 & 8 \\
2 & 3 & 4 & 5 \\
\end{array}
\]

\[
\begin{array}{cccc}
2 & 3 & 4 & 5 \\
3 & 4 & 5 & 6 \\
4 & 5 & 6 & 7 \\
5 & 6 & 7 & 8 \\
\end{array}
\]
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
Threads in OpenMP

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];
```
void saxpy(int n,
   float a,
   float *x,
   float *restrict y)
{
    #pragma acc kernels
    for (int i = 0; i < n; ++i)
       y[i] = a*x[i] + y[i];
}

... // Somewhere in main
// call SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
...

subroutine saxpy(n, a, x, y)
   real :: x(:), y(:), a
   integer :: n, i
   !$acc kernels
   do i=1,n
      y(i) = a*x(i)+y(i)
   enddo
   !$acc end kernels
end subroutine saxpy

... From main program
$ call SAXPY on 1M elements
saxpy(2**20, 2.0, x_d, y_d)
...
// Host code
int main(int argc, char** argv)
{
    // Allocate input vectors in host memory
    h_A = (float*)malloc(size);
    if (h_A == 0) Cleanup();
    h_B = (float*)malloc(size);
    if (h_B == 0) Cleanup();
    h_C = (float*)malloc(size);
    if (h_C == 0) Cleanup();

    // Initialize input vectors
    Init(h_A, N);
    Init(h_B, N);

    // Allocate vectors in device memory
    cudaMalloc((void**)&d_A, size);
    cudaMalloc((void**)&d_B, size);
    cudaMalloc((void**)&d_C, size);

    // Copy vectors to device memory
    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    // Run kernel
    int threadsPerBlock = 256;
    int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;
    VecAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);

    // Copy results from device memory to host memory
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
}

// GPU Code
__global__ void VecAdd(const float* A, const float* B, float* C, int N)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}
Threads

Splits up tasks (as opposed to arrays in data parallel) such as loops amongst separate processors.

Do communication as a side effect of data loop distribution. Not an big issue on shared memory machines. Impossible on distributed memory.

Common Implementations:
- pthreads (original Unix standard)
- OpenMP
- OpenACC
- OpenCL (Khronos Group)
- DirectCompute (Microsoft)

Very C++ 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
- C++ 11, 17, 20

Pros:
1. Doesn’t perturb data structures, so can be incrementally added to existing serial codes.
2. Becoming fairly standard for compilers.

Cons:
1. Serial code left behind will be hit by Amdahl's Law
2. Forget about taking this to the next level of scalability. You can not do this on MPPs at the machine wide level.
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
- python threads are fake (due to global interpreter lock)
- 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
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++
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.
Hybrid Coding

- Problem: given the engineering constraint of a machine made up of a large collection of multi-core processors, how do we use message passing at the wide level while still taking advantage of the local shared memory?

- Similar Problem: given a large machine with accelerators on each node (GPU or MIC), how do we exploit this architecture?

- Solution: Hybrid Coding. Technically, this could be any mix of paradigms. Currently, this is likely MPI with a directive based approach mixed in.

- At the node level, you may find OpenMP or OpenACC directives most usable.

- But, one must design the MPI layer first, and then apply the OpenMP/ACC code at the node level. The reverse is rarely a viable option.
Counterintuitive: MPI vs. OpenMP on a node

It might seem obvious that, since OpenMP is created to deal with SMP code, you would ideally like to use that at the node level, even if you use MPI for big scalability across an MPP.

Very often, it turns out that the MPI-to-the-core (pun completely intended) version is faster. This indeed seems odd.

The answer is that after going to the trouble of doing a proper MPI data decomposition, you have also greatly aided the caching mechanism (by moving concurrently accessed data into different regions of memory). Hence the win.

However, if you are only interested in node-level scaling, this would be a lot of effort.
Parallel Programming in a Nutshell

Assuming you just took our workshop

- You have to spread *something* out.
- These can theoretically be many types of abstractions: work, threads, tasks, processes, data,…
- But what they *will* be is your data. And then you will use MPI, and possibly OpenMP/ACC, to operate on that data.
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 “neighborliness” matters very much.

Is Texas vs. New Jersey a good idea?
A Few Scalability Hints

- Minimize Eliminate serial sections of code
  - Only Way To Beat Amdahl’s law
- Minimize communication overhead
  - Choose algorithms that emphasize nearest neighbor communication
  - Possibly Overlap computation and communication with asynchronous communication models
- Dynamic load balancing (at least be aware of issue)
- Minimize I/O and learn how to use parallel I/O
  - Very expensive time wise, so use sparingly (and always binary)
- Choose the right language for the job!
- Plan out your code beforehand.
  - Because the above won’t just happen late in development
  - Transforming a serial code to parallel is rarely the best strategy
Summary

- Hardware drives our software options:
  - Serial boxes can’t get to petaFLOPs (let alone exaFLOPS)
  - Moore’s Law OK, but resulting power dissipation issue is the major limiting factor
  - Multiple processors are the one current end-run around this issue
  - This won’t change any time in the foreseeable future
  - So parallel programming we will go...

- Software options are many:
  - Reality has chosen a few winners
  - You have learned the important ones
In Conclusion…

OpenMP

OpenACC

MPI
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

**APGAS**

**X10, Chapel**

**Frameworks**

**Charm++**

**Functional**

**Haskell**
Appendix I

Exascale Computing Project
<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 qualifiable metal parts</td>
<td>Find, predict, and control materials and properties</td>
<td>Accurate regional impact assessments in Earth system models</td>
<td>Accelerate and translate cancer research</td>
</tr>
<tr>
<td>Next generation simulation tools for assessing nuclear weapons performance</td>
<td>High-efficiency, low-emission combustion engine and gas turbine design</td>
<td>Reliable and efficient planning of the power grid</td>
<td>Cosmological probe of the standard model of particle physics</td>
<td>Stress-resistant crop analysis and catalytic conversion of biomass-derived alcohols</td>
<td></td>
</tr>
<tr>
<td>Response to hostile threat environments and reentry conditions</td>
<td>Materials design for extreme environments of nuclear fission and fusion reactors</td>
<td>Seismic hazard risk assessment</td>
<td>Validate fundamental laws of nature</td>
<td>Metagenomics for analysis of biogeochemical cycles, climate change, environmental remediation</td>
<td></td>
</tr>
<tr>
<td>Design and commercialization of Small Modular Reactors</td>
<td>Subsurface use for carbon capture, petroleum extraction, waste disposal</td>
<td></td>
<td>Demystify origin of chemical elements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Biofuel catalyst design</td>
<td>Scale-up of clean fossil fuel combustion</td>
<td>Whole-device model of magnetically confined fusion plasmas</td>
<td>Light source-enabled analysis of protein and molecular structure and design</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
xSDK Version 0.4.0: December 2018 (even better today)

https://xsdk.info

Each xSDK member package uses or can be used with one or more xSDK packages, and the connecting interface is regularly tested for regressions.

xSDK functionality, Dec 2018
Tested on key machines at ALCF, NERSC, OLCF, also Linux, Mac OS X

Multiphysics Application C
Application A
Application B

Alquimia
SLEPc
PETSc
hypre
SUNDIALS
AMReX
Omega_h
deal.II
More libraries

HDF5
BLAS
More external software

PFLOTTRAN
SuperLU
Strumpack
Trilinos
MFEM
DTK
Tasmanian
MAGMA
PUMI
PHIST
PLASMA

More domain components

December 2018
• 17 math libraries
• 2 domain components
• 16 mandatory xSDK community policies
• Spack xSDK installer

Domain components
• Reacting flow, etc.
• Reusable.

Libraries
• Solvers, etc.
• Interoperable.

Frameworks & tools
• Doc generators.
• Test, build framework.

SW engineering
• Productivity tools.
• Models, processes.

Impact: Improved code quality, usability, access, sustainability

Foundation for work on performance portability, deeper levels of package interoperability
The planned ECP ST SDKs will span all technology areas.
Appendix II

Endless Exascale apps...
CEED is targeting several ECP applications:

- Compressible flow (MARBL)
- Climate (E3SM)
- Urban systems (Urban)
- Subsurface (GEOS)
- Modular Nuclear Reactors (ExaSMR)
- Wind Energy (ExaWind)
- Additive Manufacturing (ExaAM)
- Combustion (Nek5000)
- Magnetic Fusion (WDMApp)

PI: Tzanio Kolev (LLNL)
ECP’s Adaptive Mesh Refinement Co-Design Center: AMReX

- Develop and deploy software to support block-structured adaptive mesh refinement on exascale architectures
  - Core AMR functionality
  - Particles coupled to AMR meshes
  - Embedded boundary (EB) representation of complex geometry
  - Linear solvers
  - Supports two modalities of use
    - Library support for AMR
    - Framework for constructing AMR applications
- Provide direct support to ECP applications that need AMR for their application
- Evaluate software technologies and integrate with AMReX when appropriate
- Interact with hardware technologies / vendors

PI: John Bell (LBNL)
ECP’s Co-Design Center for Online Data Analysis and Reduction (CODAR)

**Goal:** Replace the activities in HPC workflow that have been mediated through file I/O with in-situ methods / workflows. Data reduction, analysis, code coupling, aggregation (e.g. parameter studies).

**Cross-cutting tools:**
- Workflow setup, manager (Cheetah, Savanna); Data coupler (ADIOS-SST); Compression methods (MGARD, FTK, SZ), compression checker (Z-checker)
- Performance tools (TAU, Chimbuco, SOSFlow)

PI: Ian Foster (ANL)
**ECP’s Co-Design Center for Particle Applications: CoPA**

**Goal:** Develop algorithms and software for particle methods,

**Cross-cutting capabilities:**

- Specialized solvers for quantum molecular dynamics (Progress / BML).
- Performance-portable libraries for classical particle methods in MD, PDE (Cabana).
- FFT-based Poisson solvers for long-range forces.

**Technical approach:**

- High-level C++ APIs, plus a Fortran interface (Cabana).
- Leverage existing / planned FFT software.
- Extensive use of miniapps / proxy apps as part of the development process.

---

**PI:** Sue Mniszewski (LANL) recently replacing Tim Germann (LANL), who is taking on a larger role in ECP.
ECP’s Co-Design Center for Machine Learning: ExaLearn
Bringing together experts from 8 DOE Laboratories

- AI has the potential to accelerate scientific discovery or enable prediction in areas currently too complex for direct simulation (ML for HPC and HPC for ML)

- AI use cases of interest to ECP:
  - *Classification and regression*, including but not limited to image classification and analysis, e.g. scientific data output from DOE experimental facilities or from national security programs.
  - *Surrogate models* in high-fidelity and multiscale simulations, including uncertainty quantification and error estimation.
  - *Structure-to-function relationships*, including genome-to-phenome, the prediction of materials performance based on atomistic structures, or the prediction of performance margins based on manufacturing data.
  - *Control systems*, e.g., for wind plants, nuclear power plants, experimental steering and autonomous vehicles.
  - *Inverse problems* and optimization. This area would include, for example, inverse imaging and materials design.

- Areas in need of research
  - Data quality and statistics
  - Learning algorithms
  - Physics-Informed AI
  - Verification and Validation
  - Performance and scalability
  - Workflow and deployment

**Expected Work Product: A Toolset That . . .**

- Has a line-of-sight to exascale computing, e.g. through using exascale platforms directly, or providing essential components to an exascale workflow
- Does not replicate capabilities easily obtainable from existing, widely-available packages
- Builds in domain knowledge where possible “Physics”-based ML and AI
- Quantifies uncertainty in predictive capacity
- Is interpretable
- Is reproducible
- Tracks provenance

PI: Frank Alexander (BNL)
Machine Learning in the Light Source Workflow

Beam Line Control and Data Acquisition (DAQ)
- Data TB/s
- DAQ
- ML to design light source beam lines
- ML to control the beam line parameters
- ML at DAQ to control data as it is acquired

Local Systems
- Compressor Nodes
- Online Monitoring and Fast Feedback
- Data
- Model
- ML for data compression (e.g., hit finding). Use models learned remotely.
- ML for fast analysis at the experimental facility. Uses models learned remotely.

Network
- Model
- 10 GB/s - 1 Tb/s
- ML networks for image classification, feature detection and solving inverse problems (how to change experiment params to get desired experiment result)

Remote Exascale HPC
- Exascale Supercomputer
- Simulate experiments, beam line control and diffraction images at scale to create data for training

PI: Frank Alexander (BNL)
Exascale apps can deliver transformative products and solutions

### ExaWind
**Turbine Wind Plant Efficiency**
(Mike Sprague, NREL)
- Harden wind plant design and layout against energy loss susceptibility
- Increase penetration of wind energy

*Challenges:* linear solver perf in strong scale limit; manipulation of large meshes; overset of structured & unstructured grids; communication-avoiding linear solvers

### ExaAM
**Additive Manufacturing (AM) of Qualifiable Metal Parts**
(John Turner, ORNL)
- Accelerate the widespread adoption of AM by enabling routine fabrication of qualifiable metal parts

*Challenges:* capturing unresolved physics; multi-grid linear solver performance; coupled physics

### EQSIM
**Earthquake Hazard Risk Assessment**
(David McCallen, LBNL)
- Replace conservative and costly earthquake retrofits with safe purpose-fit retrofits and designs

*Challenges:* full waveform inversion algorithms
EQSIM: Understanding and predicting earthquake phenomenon

- **Site ground motions**
- **Vertical motion**
- **Horizontal motion**
- **Surface waves**
- **Body waves**
- **Site**
- **Path**
- **Source**

Ground motions tend to be very site specific

PI: David McCallen (LBNL)
EQSIM: The Exascale “Big Lift” – regional ground motion simulations at engineering frequencies

- Pipelines
- Long-span Bridges
- Tall Buildings
- Low-rise Buildings and Industrial Facilities
- Energy System Components
- Nuclear Power Equipment

Exascale objective

Range of historical ground motion simulations

Doubling the frequency resolution = 16X computational effort!

PI: David McCallen (LBNL)
### Exascale apps can deliver transformative products and solutions

<table>
<thead>
<tr>
<th>MFIX-Exa</th>
<th>GAMESS</th>
<th>EXAALT</th>
</tr>
</thead>
</table>
| **Scale-up of Clean Fossil Fuel Combustion** *(Madhava Syamlal, NETL)*  
- Commercial-scale demonstration of transformational energy technologies – curbing CO₂ emissions at fossil fuel power plants by 2030  
*Challenges*: load balancing; strong scaling thru transients | **Biofuel Catalyst Design** *(Mark Gordon, Ames)*  
- Design more robust and selective catalysts orders of magnitude more efficient at temperatures hundreds of degrees lower  
*Challenges*: weak scaling of overall problem; on-node performance of molecular fragments | **Materials for Extreme Environments** *(Danny Perez, LANL)*  
- Simultaneously address time, length, and accuracy requirements for predictive microstructural evolution of materials  
*Challenges*: SNAP kernel efficiency on accelerators; efficiency of DFTB application on accelerators |
### Exascale apps can deliver transformative products and solutions

<table>
<thead>
<tr>
<th>ExaSMR</th>
<th>Subsurface</th>
<th>QMCPACK</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design and Commercialization of Small Modular Reactors</strong> (Steve Hamilton, ORNL)</td>
<td><strong>Carbon Capture, Fossil Fuel Extraction, Waste Disposal</strong> (Carl Steefel, LBNL)</td>
<td><strong>Materials for Extreme Environments</strong> (Paul Kent, ORNL)</td>
</tr>
<tr>
<td>- Virtual test reactor for advanced designs via experimental-quality simulations of reactor behavior</td>
<td>- Reliably guide safe long-term consequential decisions about storage, sequestration, and exploration</td>
<td>- Find, predict and control materials and properties at the quantum level with unprecedented accuracy for the design novel materials that rely on metal to insulator transitions for high performance electronics, sensing, storage</td>
</tr>
</tbody>
</table>

**Challenges:**
- existing GPU-based MC algorithms require rework for hardware less performant for latency-bound algorithms with thread divergence; performance portability with OCCA & OpenACC not achievable; insufficient node memory for adequate CFD + MC coupling
- performance of Lagrangian geomechanics; adequacy of Lagrangian crack mechanics) + Eulerian (reaction, advection, diffusion) models; parallel HDF5 for coupling

**Challenges:**
- minimizing on-node memory usage; parallel on-node performance of Markov-chain Monte Carlo
Exascale apps can deliver transformative products and solutions

**ExaSGD**

Reliable and Efficient Planning of the Power Grid  
(Henry Huang, PNNL)

- Optimize power grid planning, operation, control and improve reliability and efficiency

*Challenges*: parallel performance of nonlinear optimization based on discrete algebraic equations and possible mixed-integer programming

---

**Combustion-PELE**

High-Efficiency, Low-Emission Combustion Engine Design  
(Jackie Chen, SNL)

- Reduce or eliminate current cut-and-try approaches for combustion system design

*Challenges*: performance of chemistry ODE integration on accelerated architectures; linear solver performance for low-Mach algorithm; explicit LES/DNS algorithm not stable
### E3SM-MMF

**Accurate Regional Impact Assessment in Earth Systems**
(Mark Taylor, SNL)

- Forecast water resources and severe weather with increased confidence; address food supply changes

**Challenges:** MMF approach for cloud-resolving model has large biases; adequacy of Fortran MPI+OpenMP for some architectures; Support for OpenMP and OpenACC

### NWChemEx

**Catalytic Conversion of Biomass-Derived Alcohols**
(Thom Dunning, PNNL)

- Develop new optimal catalysts while changing the current design processes that remain costly, time consuming, and dominated by trial-and-error

**Challenges:** computation of energy gradients for coupled-cluster implementation; on- and off-node performance

### ExaBiome

**Metagenomics for Analysis of Biogeochemical Cycles**
(Kathy Yelick, LBNL)

- Discover knowledge useful for environmental remediation and the manufacture of novel chemicals and medicines

**Challenges:** Inability of message injection rates to keep up with core counts; efficient and performant implementation of UPC, UPC++, GASNet; GPU performance; I/O performance

---

Exascale apps can deliver transformative products and solutions
E3SM-Multiscale Modeling Framework (MMF)
Cloud Resolving Climate Model for E3SM

- Develop capability to assess regional impacts of climate change on the water cycle that directly affect the US economy such as agriculture and energy production.

- Cloud resolving climate model is needed to reduce major systematic errors in climate simulations due to structural uncertainty in numerical treatments of convection – such as convective storm systems

- Challenge: cloud resolving climate model using traditional approaches requires *zettascale* resources

- E3SM “conventional” approach:
  - Run the E3SM model with a global cloud resolving atmosphere and eddy resolving ocean.
    - 3 km atmosphere/land (7B grid points) and 15-5 km ocean/ice (1B grid points)
  - Achieve throughput rate of 5 SYPD to perform climate simulation campaigns including a 500 year control simulation
  - Detailed benchmarks on KNL and v100 GPUs show negligible speedups compared to conventional CPUs
    - Low arithmetic intensity of most of the code; throughput requirements lead to strong scaling and low work per node.

- E3SM-MMF: Use a multiscale approach ideal for new architectures to achieve cloud resolving convection on Exascale
  - Exascale will make “conventional” cloud resolving simulations routine for shorter simulations (process studies, weather prediction)
  - For cloud resolving climate simulations, we need fundamentally new approaches to take advantage of exascale resources

Convective storm system nearing the Chicago metropolitan area
http://www.spc.noaa.gov/miac/AbtDerechos/derechofacts.htm

PI: Mark Taylor (SNL)
Exascale apps can deliver transformative products and solutions

**ExaSky**

Cosmological Probe of the Standard Model of Particle Physics (Salman Habib, ANL)

- Unravel key unknowns in the dynamics of the Universe: dark energy, dark matter, and inflation

Challenges: subgrid model accuracy; OpenMP performance on GPUs; file system stability and availability

**LatticeQCD**

Validate Fundamental Laws of Nature (Andreas Kronfeld, FNAL)

- Correct light quark masses; properties of light nuclei from first principles; <1% uncertainty in simple quantities

Challenges: performance of critical slowing down; reducing network traffic to reduce system interconnect contention; strong scaling performance to mitigate reliance on checkpointing

**WarpX**

Plasma Wakefield Accelerator Design (Jean-Luc Vay, LBNL)

- Virtual design of 100-stage 1 TeV collider; dramatically cut accelerator size and design cost

Challenges: scaling of Maxwell FFT-based solver; maintaining efficiency of large timestep algorithm; load balancing
High-Fidelity Whole Device Modeling of Magnetically Confined Fusion Plasmas (Amitava Bhattacharjee, PPPL)
- Prepare for ITER exps and increase ROI of validation data and understanding
- Prepare for beyond-ITER devices

**Challenges:** robust, accurate, and efficient code-coupling algorithm; reduction in memory and I/O usage

Demystify Origin of Chemical Elements (Dan Kasen, LBNL)
- What is the origin of the elements?
- How does matter behave at extreme densities?
- What are the sources of gravity waves?

**Challenges:** delivering performance on accelerators; delivering fidelity for general relativity implementation

Light Source-Enabled Analysis of Protein and Molecular Structure and Design (Amadeo Perazzo, SLAC)
- Process data without beam time loss
- Determine nanoparticle size and shape changes
- Engineer functional properties in biology and materials science

**Challenges:** improving the strong scaling (one event processed over many cores) of compute-intensive algorithms (ray tracing, M-TIP) on accelerators

Accelerate and Translate Cancer Research (Rick Stevens, ANL)
- Develop predictive preclinical models and accelerate diagnostic and targeted therapy through predicting mechanisms of RAS/RAF driven cancers

**Challenges:** increasing accelerator utilization for model search; effectively exploiting HP16; preparing for any data management or communication bottlenecks