Directive-Based Parallel Programming at Scale?

Barbara Chapman
Stony Brook University
Brookhaven National Laboratory

Charm++ Workshop, April 19 2016

http://www.cs.uh.edu/~hpctools
Agenda

• Directives: A little (pre)history
• Evolving the standard
• Today’s challenges
• Where to next?
Symmetric Multiprocessors

- 1980s saw attempts to build parallel computers with shared memory
  - Alliant
  - Sequent
  - Encore, ...
- Programmed using Fortran
  - Vendor extensions, mainly to parallelize loops
- Attempt to develop standard API
  - PCF features for loop parallelism in Fortran code
  - Fortran standards subcommittee formed

BBN Butterfly
- Every CPU able to access memory associated with other CPUs
- Big penalty for non-local access
- 15 times slower than local memory access
PCF Example

PARALLEL SECTIONS
SECTION
PARALLEL DO  I = i, N
  A(I) = B(I) * C(I)
END DO
SECTION
PARALLEL DO  J = i, M
  D(J) = F(J) / E(J)
END DO
END PARALLEL SECTIONS

• Proposed as Fortran extensions
• Team of threads execute parallel construct
• Parallel loops
• Parallel sections
• Critical, locks and post / wait
• Ordered execution

• Loop iterations distributed among threads by implementation; must be iteration-order independent
• Sections of code must be data independent
• Shared and private variables
• Loop variable undefined outside parallel loop construct
• Nested parallelism
A New Kind of Architecture, Late ‘80s
CM-5 TOP500 #1 June 1993
Using The Compute Power
High Performance Fortran (HPF)

• Directives extend Fortran for distributed memory parallel programming
  – First definition early 1993, revision 1997
  – Japanese created additional features in JA-HPF

• Main features are directives for data mapping and parallel loops
  – Work performed where the data is stored
  – Some library routines

• Broad participation in standards effort
HPF Example

!HPF$ DISTRIBUT W ( BLOCK )
!HPF$ INDEPENDENT, NEW ( X ), REDUCTION ( SUM )

DO I = 1, N
  X = W(I) * (I - 0.5)
  SUM = SUM + F ( X )
END DO

* Team of processes execute entire program
* Loop iterations are distributed among processes based on data distribution
* Communication at end of loop to obtain global value SUM

* Each process has local segment of W
* Each process has its own copy of variable X
* Each process computes local value of SUM
* SUM updated at end of loop, result replicated
What Happened to HPF?

• Compilers slow to arrive, and supported different styles of HPF programming
  – Based upon Fortran 90, also slow to mature
• Considered suitable for structured (regular) grids only
• MPI flexible and established by the time HPF compilers matured
  – Codified experience with early comms libraries
• Japanese vendors continued to add features and provide compilers after others gave up
HPF User Experience

• HPF application development was hard
  – Required global modifications
  – Incremental development not possible

• **Users had little insight into execution behavior**
  – Creation of good HPF code required insight into compilation process
  – But this was rare
  – Performance degradation could be severe

• Benefits of directive approach neither experienced nor understood by many

• Not surprisingly, **few tools** available (HPF version of Totalview was created)
MPI Becomes Widely Used

User distributes the data and computation explicitly to system processing nodes.
Return of Shared Memory

• SMPs on desktop, late 1990s (HP, Sun, Intel, IBM, ...)
  – Mainstream market, general-purpose applications
  – Mostly 2 – 4 cache coherent CPUs
  – A few bigger systems e.g. Sun’s 6400 (144 CPUs)

• Large-scale distributed shared memory (DSMs)

• Memory is distributed, but globally addressed
  – E.g. HP Exemplar, SGI Origin and Altix series
  – Looks like shared memory system to user
  – Hardware supports cache coherency
  – Origin: non-local data twice as slow
OpenMP Example

```c
 !$OMP PARALLEL DO PRIVATE ( X ), SHARED ( W )
 !$OMP& REDUCTION ( +: SUM )
    DO I = 1, N
       X = W(I) * ( I - 0.5 )
       SUM = SUM + F ( X )
    END DO
 !$OMP END PARALLEL
```

* Team of threads execute parallel region
* Loop iterations are distributed among threads
* Implicit synchronization at end of region

* All threads access same W
* Each executing thread has its own copy of variable X
* Each thread creates and initializes a private copy of shared variable SUM.
* SUM is updated at next synchronization point
Agenda

- Directives: A little (pre)history
- Evolving the standard
- Today’s challenges
- Where to next?
Cart3D OpenMP Scaling, ca. 2005

4.7 M cell mesh Space Shuttle Launch Vehicle example

- OpenMP version uses same domain decomposition strategy as MPI for data locality, avoiding false sharing and fine-grained remote data access
- OpenMP version slightly outperforms MPI version on SGI Altix 3700BX2, both close to linear scaling.

\[
M_{\infty} = 2.6 \\
\alpha = 2.09^\circ \\
\beta = 0.8^\circ
\]
Data Mapping and Affinity
Proposed OpenMP Extensions, 1999

• SGI page-based data distribution extensions
  – Allocates *pages* to memory across system nodes
  – Preserves illusion of true shared memory
• HPF-style data mappings
  – Didn’t do well on page-based system
  – SGI, Compaq

“first-touch” default mapping works pretty well (if developer is aware of it)
Omni Compiler: Cluster-enabled OpenMP, 2002

• OpenMP for a cluster (distributed memory system)
  – message passing library (MPI, PVM) provides high performance, but difficult and cumbersome.

• Use software distributed shared memory system SCASH as underlying runtime system on cluster
  – Page-based DSM
  – Related Work: OpenMP compiler for TreadMarks by Rice (later clOMP)

◆ OpenMP
  ✷ All variables are shared as defaults.
  ✷ No explicit shared memory allocation

◆ “shmemp” memory model
  ✷ All variables declared statically in global scope are private.
  ✷ The shared address space must be allocated by a library function at runtime.
  ✷ Example: SCASH, Unix “shmemp” system call
OpenMP 3.0 Introduces Tasks, 2008

- Tasks explicitly created and processed

- Each encountering thread packages a new instance of a task (code and data)

- Some thread in the team executes the task

```c
#pragma omp parallel
{
  #pragma omp single
  {
    p = listhead ;
    while (p) {
      #pragma omp task
      process (p)
      p=next (p) ;
    }
  }
}
```
Asynchronous Task Dependence

- Increase power of tasks, reduce barrier synchronization
- **Task synchronization constructs**
  - `taskwait`, and `barrier` construct

```c
int fib(int n) {
    int x, y;
    if (n < 2) return n;
    else {
        #pragma omp task shared(x)
        x = fib(n-1);
        #pragma omp task shared(y)
        y = fib(n-2);
        #pragma omp taskwait
        return x + y;
    }
}
```

- **Categories of Data Dependency**

  1. **True/Flow Dependence (RAW):**
     - Between T1 and T4 for tag 2
     - Between T1 and T4 for tag 6
     - Between T2 and T4 for tag 4
     - Between T2 and T3 for tag 10

  2. **Anti Dependence (WAR):**
     - Between T3 and T4 for tag 10 (follows program order semantics)

  3. **Output Dependence (WAW):**
     - Between T1 and T4 for tag 5 (follows program order semantics)

- `#pragma omp task depend (out: t1, t2, ...) depend (in: t4, t5)`
- Avoid the use of global locks
- Work with workstealing
- Decentralized dependency setup and resolution
Eliminating Global Barriers in Smith-Waterman

Performance in seconds for sequence size 4096 with chunk size 320

<table>
<thead>
<tr>
<th>Threads</th>
<th>OpenUH_ext</th>
<th>OmpSs</th>
<th>Quark</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.045</td>
<td>52.251</td>
<td>2.639</td>
</tr>
<tr>
<td>4</td>
<td>0.511</td>
<td>50.640</td>
<td>2.278</td>
</tr>
<tr>
<td>8</td>
<td>0.480</td>
<td>48.645</td>
<td>2.081</td>
</tr>
<tr>
<td>16</td>
<td>0.669</td>
<td>46.256</td>
<td>2.395</td>
</tr>
</tbody>
</table>

A Prototype Implementation of OpenMP Task Dependency Support; Priyanka Ghosh, Yonghong Yan, Deepak Echempati and Barbara Chapman; International Workshop on OpenMP (IWOMP) 2013
Core Heterogeneity in HPC Systems

Each node has multiple CPU cores, and some of the nodes are equipped with additional computational accelerators, such as GPUs.
OpenACC

- Directive-based programming for offloading code to accelerators
  - For Fortran, C, C++
  - Loop-based computations
- Compute directives
  - \textit{parallel}: control to the user
  - \textit{kernels}: freedom to the compiler
- Three levels of parallelism: gang, worker and vector
- Open-source and proprietary implementations
- OpenACC Validation Suite
  - C and Fortran validation for OpenACC 2.0
- SPEC Accelerator Benchmarks

\url{http://www.openacc-standard.org/}
Agenda

• Directives: A little (pre)history
• Evolving the standard
• Today’s challenges
• Where to next?
OpenACC Compiler Translation

- Need to achieve coalesced memory access on GPUs

```c
#pragma acc loop gang(2) vector(2)
for ( i = x1; i < X1; i++ ) {
    #pragma acc loop gang(3) vector(4)
    for ( j = y1; j < Y1; j++ ) { ...... }
}
```

**Double nested loop mapping.**

**Triple nested loop mapping.**

Compiling a High-level Directive-Based Programming Model for GPGPUs; Xiaonan Tian, Rengan Xu, Yonghong Yan, Zhifeng Yun, Sunita Chandrasekaran, and Barbara Chapman; 26th International Workshop on Languages and Compilers for Parallel Computing (LCPC2013)
OpenMP for Accelerators

```c
#pragma omp target data device (gpu0) map(to:n, m, omega, ax, ay, b, \ f[0:n][0:m]) map(tofrom:u[0:n][0:m]) map(alloc:uold[0:n][0:m])
while ((k<=mits)&&(error>tol))
{
  // a loop copying u[][] to uold[][] is omitted here
  ...

#pragma omp target device(gpu0)

#pragma omp parallel for private(resid,j,i) reduction(+:error)
for (i=1;i<(n-1);i++)
for (j=1;j<(m-1);j++)
{
  resid = (ax*(uold[i-1][j] + uold[i+1][j])\n           + ay*(uold[i][j-1] + uold[i][j+1]) + b * uold[i][j] - f[i][j])/b;
  u[i][j] = uold[i][j] - omega * resid;
  error = error + resid*resid ;
} // rest of the code omitted  ...
```
Dynamic Program Adaptation

- OpenMP fairly amenable to dynamic adaptation
  - Adjustment of thread count, schedule
  - Adaptive barriers, reduction routines
  - Runtime decisions
  - Tasks, mergeable
- Use of performance interface to inform dynamic tools
  - Can help adjust data layout, find memory performance problems
- Potential useful for variety of runtime techniques

![Diagram showing relationships between OpenMP Program (object code), OpenMP Runtime Library, Collector API, executable ./a.out, and Performance Tool with request and event arrows.]}
False Sharing: Monitoring Results

- Cache line invalidation measurements

<table>
<thead>
<tr>
<th>Program name</th>
<th>1-thread</th>
<th>2-threads</th>
<th>4-threads</th>
<th>8-threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram</td>
<td>13</td>
<td>7,820,000</td>
<td>16,532,800</td>
<td>5,959,190</td>
</tr>
<tr>
<td>kmeans</td>
<td>383</td>
<td>28,590</td>
<td>47,541</td>
<td>54,345</td>
</tr>
<tr>
<td>linear_regression</td>
<td>9</td>
<td>417,225,000</td>
<td>254,442,000</td>
<td>154,970,000</td>
</tr>
<tr>
<td>matrix_multiply</td>
<td>31,139</td>
<td>31,152</td>
<td>84,227</td>
<td>101,094</td>
</tr>
<tr>
<td>pca</td>
<td>44,517</td>
<td>46,757</td>
<td>80,373</td>
<td>122,288</td>
</tr>
<tr>
<td>reverse_index</td>
<td>4,284</td>
<td>89,466</td>
<td>217,884</td>
<td>590,013</td>
</tr>
<tr>
<td>string_match</td>
<td>82</td>
<td>82,503,000</td>
<td>73,178,800</td>
<td>221,882,000</td>
</tr>
<tr>
<td>word_count</td>
<td>4,877</td>
<td>6,531,793</td>
<td>18,071,086</td>
<td>68,801,742</td>
</tr>
</tbody>
</table>
False Sharing: Data Analysis Results

- Determining the variables that cause misses

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Global/static data</th>
<th>Dynamic data</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram</td>
<td>-</td>
<td>main_221</td>
</tr>
<tr>
<td>linear_regression</td>
<td>-</td>
<td>main_155</td>
</tr>
<tr>
<td>reverse_index</td>
<td>use_len</td>
<td>main_519</td>
</tr>
<tr>
<td>string_match</td>
<td>key2_final</td>
<td>string_match_match_map_26</td>
</tr>
<tr>
<td>word_count</td>
<td>length, use_len,</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>words</td>
<td></td>
</tr>
</tbody>
</table>
Runtime False Sharing Detection

Energy Management Tools

• OpenMP runtime settings can be adjusted statically and dynamically for best performance
  – Number of threads, scheduling policy and chunk size, wait policy, binding policy, may all affect performance

• Selections are not independent of power cap

• Modeling may help select settings to optimize both energy and execution performance

%age improvement in Co-MD application under different power capping
Agenda

• Directives: A little (pre)history
• Evolving the standard
• Today’s challenges
• Where to next?
Memory Will Change
So Will Integration of Accelerators

HCA, CAPI, GPU interconnect

Programmability

Diversity

Global Memory

CPU L2

GPU L2

HW Coherence

Coherence Bus

CAPP

Power8

X86, ARM64, POWER CPU

NVLink

POWER CPU
A Layered Programming Approach

- Computational Chemistry
- Climate Research
- Astrophysics

DSLs, other means for application scientists to provide information

Adapted versions of today’s portable parallel programming APIs (MPI, OpenMP, PGAS, Charm++)

Maybe some non-portable low-level APIs (threads, CUDA, Verilog)

Machine code, device-level interoperability stds, powerful runtime

- Applications
- New kinds of info
- Familiar
- Custom
- Very low-level
- Heterogeneous Hardware
More Dynamic Execution?

- What will the runtime (RT) environment look like? How dynamic will it be?
- Role of runtime system? Relationship between RT and OS, programming models? How is information exchanged?

Performance less predictable in dynamic execution environment
OpenMP in an Exascale World

- **OpenX**: prototype software stack for Exascale systems
  - HPX is runtime system
  - Lightweight threads
  - Thread migration for load balancing, throughput.
- Translating OpenMP -> HPX
  - Maps OpenMP task and data parallelism onto HPX
  - Exploit data flow execution capabilities at scale
  - Big increase in throughput for fine-grained tasks
- Migration path for OpenMP applications

---

**Diagram**

- **XPRESS Migration Stack**
- **MPI/OpenMP Application**
  - **MPI**
  - **OpenMP compiler**
    - **OpenMP Thin Runtime Glue**
  - **Legacy stack**
  - **HPX**
  - **OpenX**
OpenMP over HPX (on-going work)

- **Execution model:** dynamic adaptive resource management; message-driven computation; efficient synchronization; global name space; task scheduling

- **OpenMP translation:**
  - No direct interface to OS threads
  - No tied tasks
  - Threadprivate tricky, slow
  - Doesn’t support places
  - OpenMP task dependencies via futures
  - HPX locks faster than OS locks

---

**LU Run Time on 40 Threads, size = 8192**

- Intel
- HPX

**Graph:**
- X-axis: Block Size
- Y-axis: time (seconds)
- 512, 341, 256, 228, 171, 128, 114, 85, 64
- Run time comparison between Intel and HPX across different block sizes.
Synchronization in OpenMP Execution

A Data-Centric Era

- Continuum of needs from computation-heavy to data heavy
- Potentially within a single application or workflow
- Need to address data movement in its entirety
  - Data Layout
  - New kinds of memory
- What role does user play?
Where are Directives Headed?

• OpenMP has shown significant staying power despite some big changes in hardware characteristics
  • Broad user base; yet strong HPC representation
  • Paying more attention to data locality, affinity, tasking
• Need to continue to evolve directives and implementation
  – Data and memory challenges remain
  – Less synchronization, more tasks, is good
  – Performance; validation, power/energy savings,..
  – Runtime: resources, more dynamic execution
• What about level of abstraction?
  – Performance portability is a major challenge
  – OpenMP codes often hardwire in system-specific details
Wrap-Up

• Programmers need portable, productive programming interfaces
  – Directives help deliver new concepts
  – Hardware changes require us to continue to adapt
  – Importance of accelerator devices likely to grow
  – Many new challenges posed by diversity, large data sets, memory and new application trends

• Directives pretty successful

• Not all the answers are in the programming interface
  – New or adapted algorithms
  – Novel compiler translations; modeling for smart decisions
  – Innovative implementations and runtime adaptations
  – Tools to facilitate development and tuning