Computer Software:
The 'Trojan Horse' of HPC

Steve Wallach
swallach"at"conveycomputer "dot"com
Convey <= Convex++
Discussion

• The state of HPC software – today
• What Convey Computer is doing
• The path to Exascale Computing
• In the old days, we were told
  “Beware of Greeks bearing gifts”
Today

• “Beware of Geeks bearing Gifts”
What problems are we solving

• New Hardware Paradigms
• Uniprocessor Performance leveling
• MPP and multi-threading for the masses
Deja Vu

- Multi-Core Evolves
  - Many Core
  - ILP fizzes
- x86 extended with sse2, sse3, and sse4
  - application specific enhancements
  - Co-processor within x86 micro-architecture
- Basically performance enhancements by
  - On chip parallel
  - Instructions for specific application acceleration
    - One application instruction replaces MANY generic instructions
- Déjà vu – all over again – 1980’s
  - Need more performance than micro
  - GPU, CELL, and FPGA’s
    - Different software environment
- Heterogeneous Computing AGAIN
Current Languages

- Fortran 66 ➞ Fortran 77 ➞ Fortran 95 ➞ 2003
  - HPC Fortran
  - Co-Array Fortran
- C ➞ C++
  - UPC
  - Stream C
  - C# (Microsoft)
  - Ct (Intel)
Another Bump in the Road

• GPGPU’s are very cost effective for many applications.

• Matrix Multiply
  – Fortran

```fortran
  do i = 1,n1
    do k = 1,n3
      c(i,k) = 0.0
    enddo
    do j = 1,n2
      c(i,k) = c(i,k) + a(i,j) * b(j,k)
    enddo
  enddo
```

Simplified Matrix Multiplication in CUDA, Using Tiled Algorithm

```c
__global__ void matmulKernel( float* C, float* A, float* B, int N2, int N3 )
{
    int bx = blockIdx.x, by = blockIdx.y;
    int tx = threadIdx.x, ty = threadIdx.y;
    int aFirst = 16 * by * N2;
    int bFirst = 16 * bx;
    float Csub = 0;
    for( int j = 0; j < N2; j += 16 ) {
        __shared__ float Atile[16][16], Btile[16][16];
        Atile[ty][tx] = A[aFirst + j + N2 * ty + tx];
        Btile[ty][tx] = B[bFirst + j*N3 + b + N3 * ty + tx];
        __syncthreads();
        for( int k = 0; k < 16; ++k )
            Csub += Atile[ty][k] * Btile[k][tx];
        __syncthreads();
    }
    int c = N3 * 16 * by + 16 * bx;
    C[c + N3 * ty + tx] = Csub;
}
```

doctor

Pornographic Programming:
*Can’t define it, but you know When you see it.*

http://www.linuxjournal.com/article/10216

Michael Wolfe – Portland Group

How We Should Program GPGPUs November 1st, 2008
Accelerators can be beneficial. It isn’t “free” (like waiting for the next clock speed boost)

- Worst case - you will have to completely rethink your algorithms and/or data structures
- Performance tuning is still time consuming
- Don’t forget our long history of parallel computing...

Fall Creek Falls Conference, Chattanooga, TN - Sept. 2009
One of These Things Isn’t Like the Other...Now What? Pat McCormick, LANL
• Tradeoff programmer productivity vs. performance

• Web programming is mostly done with scripting and interpretive languages
  – Java
  – JavaScript

• Server-side programming languages (Python, Ruby, etc.).

• Matlab users tradeoff productivity for performance
  – Moore’s Law helps performance
  – Moore’s Law hurts productivity

• Multi-core

• What languages are being used

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>=</td>
<td>Java</td>
<td>18.033%</td>
<td>-2.11%</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>=</td>
<td>C</td>
<td>17.809%</td>
<td>+1.03%</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>=</td>
<td>C++</td>
<td>10.757%</td>
<td>+0.16%</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>=</td>
<td>PHP</td>
<td>8.934%</td>
<td>-0.74%</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>=</td>
<td>(Visual) Basic</td>
<td>5.868%</td>
<td>-2.07%</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>↑</td>
<td>C#</td>
<td>5.196%</td>
<td>+0.66%</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>↓</td>
<td>Python</td>
<td>4.266%</td>
<td>-0.49%</td>
<td>A</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>↑</td>
<td>Perl</td>
<td>3.200%</td>
<td>-0.71%</td>
<td>A</td>
</tr>
<tr>
<td>9</td>
<td>45</td>
<td>↑</td>
<td>Objective-C</td>
<td>2.469%</td>
<td>+2.35%</td>
<td>A</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>↑</td>
<td>Delphi</td>
<td>2.394%</td>
<td>+0.21%</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>↓</td>
<td>JavaScript</td>
<td>2.191%</td>
<td>-1.83%</td>
<td>A</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>↓</td>
<td>Ruby</td>
<td>2.070%</td>
<td>-0.56%</td>
<td>A</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>↓</td>
<td>FL/SQL</td>
<td>0.787%</td>
<td>-0.09%</td>
<td>A</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>=</td>
<td>SAS</td>
<td>0.703%</td>
<td>-0.06%</td>
<td>A</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>=</td>
<td>Pascal</td>
<td>0.702%</td>
<td>-0.06%</td>
<td>A-</td>
</tr>
<tr>
<td>16</td>
<td>18</td>
<td>↑</td>
<td>Lisp/Scheme/Clojure</td>
<td>0.654%</td>
<td>+0.05%</td>
<td>B</td>
</tr>
<tr>
<td>17</td>
<td>19</td>
<td>↑</td>
<td>Lua</td>
<td>0.592%</td>
<td>+0.04%</td>
<td>B</td>
</tr>
<tr>
<td>18</td>
<td>20</td>
<td>↑</td>
<td>MATLAB</td>
<td>0.589%</td>
<td>+0.06%</td>
<td>B</td>
</tr>
<tr>
<td>19</td>
<td>16</td>
<td>↓</td>
<td>ABAP</td>
<td>0.577%</td>
<td>-0.15%</td>
<td>B</td>
</tr>
<tr>
<td>20</td>
<td>27</td>
<td>↑</td>
<td>PowerShell</td>
<td>0.529%</td>
<td>+0.23%</td>
<td>B</td>
</tr>
</tbody>
</table>
Interpretive Languages

- User cycles are more important than computer cycles
  - We need ISA’s that model and directly support these interpretive languages
  - FPGA’s cores to support extensions to the standard ISA

- Language Directed Design
  - ACM-IEEE Symp. on High-Level-Language Computer Architecture, 7-8 November 1973, College Park, MD
  - FAP – Fortran Assembly Program (IBM 7090)
  - B1700 – S-languages (Burroughs)
    - Architecture defined by compiler writers
Cautionary Tale: A Brief History of Languages

- **When vector machines were king**
  - Parallel “languages” were loop annotations (IVDEP)
  - Performance was fragile, but there was good user support

Kathy Yelick, 2008 Keynote, Salishan Conference
Introduction to Convey Computer

- Developer of the HC-1 hybrid-core computer system
- Leverages Intel x86 ecosystem
- FPGA based coprocessor for performance & efficiency
- Experienced team
The Convey Hybrid-Core Computer

- Extends x86 ISA with performance of a hardware-based architecture
- Adapts to application workloads
- Programmed in ANSI standard C/C++ and Fortran
- Leverages x86 ecosystem
Convey Product

- Reconfigurable Co-Processor to Intel x86-64
- Shared 64-bit Virtual and Physical Memory (cache coherent)
- Coprocessor executes instructions that are viewed as extensions to the x86 ISA
- Convey Developed Compilers: C, C++, & Fortran based on open 64)
  - Automatic Vectorization/Parallelization
  - SIMD Multi-threading
  - Generates both x86 and coprocessor instructions

![Diagram of Convey Product architecture]

- Cache coherent
- Shared virtual and Physical memory

Intel x86_64
Linux ecosystem

PCIe

delivered application specific performance

Coprocessor
host interface
Application Engines
coherent memory controller

4 DIMM channels
16 DIMM channels (80GB/sec)
ECC memory
Inside the Coprocessor

Personalities dynamically loaded into AEs implement application specific instructions

System interface and memory management implemented by coprocessor infrastructure

31 way interleaving

16 DDR2 memory channels
Standard or Scatter-Gather DIMMs
80GB/sec throughput

Non-blocking
Virtual output queuing
Round-robin arbitration

swallach - oct 2010 - ucsb
Convey Scatter-Gather DIMMs

• Standard DIMMs are optimized for cache line transfers
  – performance drops dramatically when access pattern is strided or random
• Convey Scatter-Gather DIMMs are optimized for 8-byte transfers
  – deliver high bandwidth for random or strided 64-bit accesses
  – prime number (31) interleave maintains performance for power-of-two strides
  – Supports both SIMD and Parallel multi-threading compute model
  – Out of order loads and stores
Strided Memory

Strided Memory Access Performance

- Convey memory system optimized for 64-bit memory access
- High bandwidth for all strides except multiples of 31

```c
for( i=0; i<n*incx; i+=incx ) {
    yy[i] += t*xx[i];
}
```

- Measured with strid3.c written by Mark Seager
- Modified to use long long for SP vector personality

1.2 GBytes/sec
SP Personality

4 Application Engines / 32 Function Pipes
vector elements distributed across function pipes

Vector architecture optimized for Signal Processing applications

Load-store vector architecture with single precision floating point

Multiple function pipes for data parallelism

Multiple functional units and out-of-order execution for instruction parallelism
UCSD InsPect Personality

Hardware implementation of search kernel from InsPect proteomics application

1-of-40 State Machines

Entire “Best Match” routine implemented as state machine

Multiple state machines for data parallelism

Operates on main memory using virtual addresses

MIMD mode of computation

Lightweight thread dispatcher on x86
Convey Linux Kernel

Shared Libraries
- Coprocessor allocation
- Coprocessor dispatch & synchronization

Loadable Modules
- Initialization & management
- Exception and error handling
- Context save/restore

Enhanced Linux Kernel
- Multiple page sizes
- Page coloring for prime number interleave
- Process mgmt for coprocessor threads
- Intel® 64/Linux LSB binary compatibility

Shared Libraries
- libcny_usr.so
- libcny_sys.so

Kernel & Loadable Modules
- cpSYS
- cpHAL
- Linux Kernel
- cpKERN

Intel Host Processor
Convey Coprocessor
Berkeley’s 13 Motifs

“Motif/Dwarf” Popularity
(\textbf{Red Hot} $\rightarrow$ \textbf{Blue Cool})

- How do compelling apps relate to 13 motif/dwarfs?

http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-23.html
What does this all mean?

How do we create architectures to do this?

Each application is a different point on this 3D grid (actually a curve)

What does this all mean?

Each application is a different point on this 3D grid (actually a curve)

How do we create architectures to do this?

What does this all mean?

Each application is a different point on this 3D grid (actually a curve)

How do we create architectures to do this?
Convey Compilers

- Program in ANSI standard C/C++ and Fortran
- Unified compiler generates x86 & coprocessor instructions
- Seamless debugging environment for Intel & coprocessor code
- Executable can run on x86_64 nodes or on Convey Hybrid-Core nodes
Using Personalities

- Program using ANSI standard C/ C++ and Fortran
- User specifies personality at compile time
- OS demand loads personalities at runtime

C/C++  Fortran

Convey Software Development Suite

Hybrid-Core Executable x86-64 and Coprocessor Instructions

Personalities

Convey HC-1

Intel x86  Coprocessor

description file specifies available instructions

personality loaded at runtime by OS
3D Finite Difference (3DFD) Personality

- designed for nearest neighbor operations on structured grids
  - maximizes data reuse
- reconfigurable “registers”
  - 1D (vector), 2D, and 3D modes
  - 8192 elements in a register
- operations on entire cubes
  - “add points to their neighbor to the left times a scalar” is a single instruction
  - up to 7 points away in any direction
- finite difference method for post-stack reverse-time migration

\[
x(I,J,K) = S0*Y(I,J,K) + S1*Y(I-1,J,K) + S2*Y(I+1,J,K) + S3*Y(I,J-1,K) + S4*Y(I,J+1,K) + S5*Y(I,J,K-1) + S6*Y(I,J,K+1)
\]
3D Grid Data Storage

- Personality maps 3-D grid data structure to register files within function units to achieve maximum data reuse.
- Dimensions of 3-D data structure can be easily changed.
- A few dozen instructions will perform a 14\textsuperscript{th} order stencil operation.
3DFD Personality Framework

- The number of Function Pipe Groups is set to match available memory bandwidth.
- The number of Function Pipes is set based on the resources available.
- 3-D HW grid structure
  - X axis, Number of FPG
  - Y axis, Number of FP per FPG
  - Z axis, Vector Elements within a FP
Technology Roadmap

- Improved hardware and software reliability
- New I/O technology
- 3D chip-level integration
- New programming model
- Improved interconnect technology
- Demonstrate > 3X power efficiency gain
- SW scalability to 100M threads
- Latency tolerant algorithms
- Demonstrate > 3X power efficiency gain over 2015
- 10X memory BW
- Application scalability to 1TB threads
- Improved resilience through local recovery and migration
- Exascale Science


100 Peta

1 Exa

Net Throughput

DOE Exascale Initiative Technical Roadmap
Predictions and Conclusions

- Parallelism will explode
  - Number of cores will double every ~2 years
  - Petaflop (million processor) machines will be common in HPC by 2015 (all top 500 machines will have this)
- Performance will become a software problem
  - Parallelism and locality are fundamental; can save power by pushing these to software
- Locality will continue to be important
  - On-chip to off-chip as well as node to node
  - Need to design algorithms for what counts (communication not computation)
- Massive parallelism required (including pipelining and overlap)

Kathy Yelick, 2008 Keynote, Salishan Conference
Concluding

• Uniprocessor Performance has to be increased
  – Heterogeneous here to stay
  – The easiest to program will be the correct technology
• Smarter Memory Systems (PIM)
• New HPC Software must be developed.
  – SMARTER COMPILERS
  – ARCHITECTURE TRANSPARENT
• New algorithms, not necessarily new languages
Finally

AS LEAD SOFTWARE ENGINEER, I GIVE YOU THE FIRST UNIT OF OUR TEN THOUSAND COPY PRODUCTION RUN.