OpenMP 4.X to programming HPC applications on heterogeneous architectures?

Thierry Gautier
thierry.gautier@inrialpes.fr
MOAIS, INRIA, Grenoble
Outline

• Introduction
• OpenMP 3.1
• Improving overhead in OpenMP task management
• Extending loop scheduler
• ADT K’STAR & OpenMP4.X
• Conclusions
Parallel heterogeneous architecture

- Complex architecture
  - Computing resources
    - SIMD Units, CPU Core, GPU, Intel Xeon Phi,..., MPPA, FPGA...
  - Memory
    - hierarchical memory
      - registers, cache L1, L2, L3, local memory bank, global memory, remote memory
    - private / shared cache
  - Interconnection networks
    - between cores & memory = memory network (HyperTransport, QuickPath, PCI, ...)
    - between machines (Ethernet, InfiniBand,...)

- High complexity
  - million of components
  - heterogeneous
    - memory access
    - computing capability
Role of a parallel programming model

• Abstraction of the hardware
  ‣ Hide complexity
    ◦ Numerous computing resources
      - SIMD Units, Core
    ◦ NUMA (heterogeneous) architecture
      - Latency + contention in “remote” memory access
  ‣ Good compromise performance/simplicity

• Supercomputer = high performance network + multicores + accelerators
  ‣ Network: MPI
  ‣ Multicores: OpenMP
  ‣ Accelerators: Cuda, OpenCL, OpenACC
  ‣ SIMD Units: Compiler or extension

  } \[\text{OpenMP 4.X ?}\]

• Main Difficulty = scheduling
  ‣ Scheduling is a hard problem
    ◦ heterogenous architecture
      - 1 GPU may be about 100 times faster than 1 CPU core
      - Taking into account communication

• Which programming model ?
  ‣ Currently at least 3 / 4 different models...
OpenMP

• Explicit Parallelism
  ‣ Code annotation for Fortran, C, C++

• OpenMP 3.1
  ‣ Task parallelism
    ◦ #pragma omp task [...] 
    ◦ Creation of task is a non blocking operation
      - completion guaranteed by synchronization
    ◦ Task execution may be concurrent with the thread that creates it
  ‣ Data parallelism = parallel loop
    ◦ #pragma omp parallel for / #pragma omp for in a parallel region

• OpenMP 4.X
  ‣ SIMD Units
    ◦ Public beta, SC 2012
  ‣ Dependencies between tasks
  ‣ Accelerator
    ◦ ~ OpenACC: Offloading code to accelerator
Some Limitations

• Programming Model
  ‣ Reduction operators are assumed to be associative & commutative
  ‣ Parallel loop is only for independent loop
    ○ automatic parallelization has some limitation
  ‣ Scheduling parallel loop: static, dynamic, guided
    ○ "chunk size" decision!
  ‣ Becomes more and more complex...

• Runtime support
  ‣ Overhead in task management
    ○ Intel Compiler has lower overhead over GCC. GCC does not work with lot of fine grain tasks.
  ‣ Internal algorithms (scheduling, termination, ...)
    ○ may not scale well on NUMA

• OpenMP Standard evolves slowly but it does!

• Extensions
  1. Improving overhead in task management
  2. New NUMA aware loop schedule for irregular iterations
  3. Future work around OpenMP4.X features
Improving OpenMP task implementation

- [IWOMP 2012]
- Barcelona OpenMP Task Suite
  - Using libKOMP = our libGOMP implementation (on top of XKaapi)
  - A set of representative benchmarks to evaluate OpenMP tasks implementations

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments used</th>
<th>Domain</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment</td>
<td>prot100.aa</td>
<td>Dynamic programming</td>
<td>Aligns sequences of proteins</td>
</tr>
<tr>
<td>FFT</td>
<td>n=33,554,432</td>
<td>Spectral method</td>
<td>Computes a Fast Fourier Transformation</td>
</tr>
<tr>
<td>Floorplan</td>
<td>input.20</td>
<td>Optimization</td>
<td>Computes the optimal placement of cells in a floorplan</td>
</tr>
<tr>
<td>NQueens</td>
<td>n=14</td>
<td>Search</td>
<td>Finds solutions of the N Queens problem</td>
</tr>
<tr>
<td>MultiSort</td>
<td>n=33,554,432</td>
<td>Integer sorting</td>
<td>Uses a mixture of sorting algorithms to sort a vector</td>
</tr>
<tr>
<td>SparseLU</td>
<td>n=128 m=64</td>
<td>Sparse linear algebra</td>
<td>Computes the LU factorization of a sparse matrix</td>
</tr>
<tr>
<td>Strassen</td>
<td>n=8192</td>
<td>Dense linear algebra</td>
<td>Computes a matrix multiply with Strassen’s method</td>
</tr>
<tr>
<td>UTS</td>
<td>medium.input</td>
<td>Search</td>
<td>Computes the number of nodes in an Unbalanced Tree</td>
</tr>
</tbody>
</table>

- Evaluation platforms
  - AMD48: 4x12 AMD Opteron (6174) cores
  - Intel32: 4x8 Intel Xeon (X7560) cores

- Softwares
  - gcc 4.6.2 + libGOMP
  - gcc 4.6.2 + libKOMP
  - icc 12.1.2 + Intel OpenMP runtime (KMP)
# Running OpenMP BOTS with libKOMP

## Speed-Up of BOTS kernels on the AMD48 platform

<table>
<thead>
<tr>
<th>kernel</th>
<th>libGOMP</th>
<th>libKOMP</th>
<th>Intel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment</td>
<td>38.8</td>
<td><strong>40.0</strong></td>
<td>37.0</td>
</tr>
<tr>
<td>FFT</td>
<td>0.5</td>
<td><strong>12.2</strong></td>
<td>12.0</td>
</tr>
<tr>
<td>Floorplan</td>
<td>27.6</td>
<td><strong>32.7</strong></td>
<td>29.2</td>
</tr>
<tr>
<td>NQueens</td>
<td>43.7</td>
<td><strong>47.8</strong></td>
<td>39.0</td>
</tr>
<tr>
<td>MultiSort</td>
<td>0.6</td>
<td><strong>13.2</strong></td>
<td>11.3</td>
</tr>
<tr>
<td>SparseLU</td>
<td>44.1</td>
<td><strong>44.4</strong></td>
<td>35.0</td>
</tr>
<tr>
<td>Strassen</td>
<td>20.8</td>
<td><strong>22.4</strong></td>
<td>20.5</td>
</tr>
<tr>
<td>UTS</td>
<td>0.9</td>
<td><strong>25.3</strong></td>
<td>15.0</td>
</tr>
</tbody>
</table>

- **Evaluation platforms**
  - AMD48: 4x12 AMD Opteron (6174) cores
  - Intel32: 4x8 Intel Xeon (X7560) cores

- **Softwares**
  - gcc 4.6.2 + libGOMP
  - gcc 4.6.2 + libKOMP
  - icc 12.1.2 + Intel OpenMP runtime (KMP)
Automatic chunk size decision

- Parallel loop
  - which OpenMP schedule: static, dynamic, guided?
  - which chunk size? depend on the application or the instance or the architecture!
    - To “big” = less parallelism, difficult to schedule
    - To “small” = overhead

  ➡ Automatic selection of the “chunk size”

- Proposed solution: new adaptive OpenMP loop scheduler
  - Runtime selection of the grain size
  - NUMA aware scheduling
  - Extension of libGOMP (runtime support of GCC)
  - [submitted to IWOMP2013]
Particle collision detection

- Iterative application
  - One step of the code ~ parallel loop (2.9e6 particles)
    - irregular computation
- GCC / OpenMP libGOMP + instrumentation
- Report activity (#ticks) per core
  - AMD 48 cores machine
  - compute (green): user code
  - schedule (red): loop initialization, load balancing, loop termination, ...
  - #iterations per core (blue curve)

Dynamic schedule

Chunk size=1
  time = 98.3ms

Chunk size=1000

Best Chunk size=3000
  time = 46ms
Particle collision detection

- 1 step of the code ~ parallel loop (2.9e6 particles)
  - irregular computation
- GCC / OpenMP libGOMP + instrumentation + new loop schedule
- Report activity (#ticks) per core
  - AMD 48 cores machine
  - compute (green): user code
  - schedule (red): loop initialization, load balancing, loop termination, ...
  - #iterations per core (blue curve)

Static & Adaptive schedule

Static OpenMP schedule
time = 12.41ms

New Adaptive schedule
time = 6.7ms
INRIA ADT K’START

• OpenMP 4.X as a good candidate for programming heterogeneous multicores
  › SIMD Units, CPUs, GPUs or Intel Xeon Phi (MIC)
  › Standard not yet publicity available but access to discussions & proposals is possible

• What’s new / under discussion ?
  › SIMD annotation
    ◢ SSE unit (Intel Xeon & Xeon Phi), Nvidia SM, Multicore / multi DSP (Texas Instrument), ...
  › Dependent tasks with data flow dependencies
  › Accelerator: integrate OpenACC proposition into OpenMP standard

• ADT K’START (~2014-2015)
  › EPI INRIA MOAIS (Grenoble, T.Gautier) and RUNTIME (Bordeaux, O. Aumage)
  › Goal first implementation of OpenMP annotation for accelerator & dependent tasks
    ◢ based on the current discussion of the standard
    ◢ Runtime will be based on StarPU and XKaapi
  › Engineer position funded
    ◢ if you have student or engineer with some experience!
    ◢ skills: compiler design, Fortran, C, C++,...
XKaapi’ compiler prototype

```c
#include <cblas.h>
#include <clapack.h>
void Cholesky( double* A, int N, size_t NB )
{
    for (size_t k=0; k < N; k += NB)
    {
        #pragma kaapi task readwrite(&A[k*N+k]{ld=N; [NB][NB]})
        clapack_dpotrf( CblasRowMajor, CblasLower, NB, &A[k*N+k], N );
        for (size_t m=k+ NB; m < N; m += NB)
        {
            #pragma kaapi task readwrite(&A[k*N+k]{ld=N; [NB][NB]})
            cblas_dtrsm ( CblasRowMajor, CblasLeft, CblasLower, CblasNoTrans, CblasUnit,
                          NB, NB, 1.0, &A[k*N+k], N, &A[m*N+k], N );
        }
        for (size_t m=k+ NB; m < N; m += NB)
        {
            #pragma kaapi task readwrite(&A[m*N+k]{ld=N; [NB][NB]})
            cblas_dsyrk ( CblasRowMajor, CblasLower, CblasNoTrans,
                          NB, NB, -1.0, &A[k*N+k], N, 1.0, &A[m*N+m], N );
            for (size_t n=k+NB; n < m; n += NB)
            {
                #pragma kaapi task readwrite(&A[m*N+n]{ld=N; [NB][NB]})
                cblas_dgemm ( CblasRowMajor, CblasNoTrans, CblasTrans,
                              NB, NB, NB, -1.0, &A[m*N+k], N, &A[n*N+k], N, 1.0, &A[m*N+n], N );
            }
        }
    }
}
```
#include <cblas.h>
#include <clapack.h>
void Cholesky( double* A, int N, size_t NB )
{
    #pragma omp parallel
    for (size_t k=0; k < N; k += NB)
    #pragma omp single
    {
        #pragma omp task depend(inout: &A[k*N+k]{ld=N; [NB][NB]})
            clapack_dpotrf( CblasRowMajor, CblasLower, NB, &A[k*N+k], N );
        
        for (size_t m=k+ NB; m < N; m += NB)
        {
            #pragma omp task depend(in: &A[k*N+k]{ld=N; [NB][NB]}, \ 
                                inout: &A[m*N+k]{ld=N; [NB][NB]})
                cblas_dtrsm ( CblasRowMajor, CblasLeft, CblasLower, CblasNoTrans, CblasUnit, 
                                NB, NB, 1., &A[k*N+k], N, &A[m*N+k], N );
        }
        
        for (size_t m=k+ NB; m < N; m += NB)
        {
            #pragma omp task depend(in: &A[m*N+k]{ld=N; [NB][NB]}, \ 
                                inout: &A[m*N+m]{ld=N; [NB][NB]})
                cblas_dsyrk ( CblasRowMajor, CblasLower, CblasNoTrans, 
                                NB, NB, -1.0, &A[m*N+k], N, 1.0, &A[m*N+m], N );
        }
        
        for (size_t n=k+NB; n < m; n += NB)
        {
            #pragma omp task depend(in: &A[m*N+k]{ld=N; [NB][NB]}, &A[n*N+k]{ld=N; [NB][NB]}, \ 
                                inout: &A[m*N+n]{ld=N; [NB][NB]})
                cblas_dgemm ( CblasRowMajor, CblasNoTrans, CblasTrans, 
                                NB, NB, NB, -1.0, &A[m*N+k], N, &A[n*N+k], N, 1.0, &A[m*N+n], N );
        }
    }
Overlapping GPU communication

- DGEMM: 1GPU
  - CUBLAS DGEMM with and without taking into account data transfers
  - XKaapi DGEMM: parallel block version

![Graph showing performance metrics]

- GPU Peak (no copy): 315 GFlop/s
- XKaapi: 309 GFlop/s
- CUBLAS with copy: 250 GFlop/s
Cholesky, multi-CPUs/multi-GPUs

Matrix order

Gflop/s

4096 8192 12288 16384 20480 24576 28672 32768 36864 40960

4CPU+8GPU 6CPU+6GPU 8CPU+4GPU 10CPU+2GPU 11CPU+1GPU
Conclusions

• **OpenMP is good opportunity for research on runtime**
  ‣ GCC libGOMP is public, easy to extend, modify
  ‣ Intel has released public version for its OpenMP runtime
  ‣ good opportunity to promote research development through Intel compiler

• **OpenMP task**
  ‣ improvement to provide fine grain implementation
  ‣ important to simplify the scheduling

• **OpenMP loop**
  ‣ existing schedulers cannot make abstraction of the chunk_size
  ‣ NUMA aware scheduling

• **INRIA ADT K’STAR**
  ‣ to promote runtime (StarPU, XKaapi) through a future OpenMP 4.X standard
Thank you for your attention!

http://kaapi.gforge.inria.fr
Comparison with OMPSS, StarPU

- DPOTRF matrix size 40960, BS 1024

No OMPSS result due to a memory problem for big matrices (bug if bigger than 10280)
Comparison with OMPSS, StarPU

- DGEMM matrix size 10240, block size 1024
Comparison with OMPSS, StarPU

- DPOTRF matrix size 10240, block size 1024
Impact of CPU number

- DPOTRF. Parallel panel factorization on x-CPPUs, 4 GPUs.

  Matrix size 40960, BS=1204

<table>
<thead>
<tr>
<th>#CPU</th>
<th>4096</th>
<th>8192</th>
<th>16384</th>
<th>32768</th>
<th>40960</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>53.85 ±0.98</td>
<td>206.38 ±2.70</td>
<td>622.55 ±7.90</td>
<td>962.21 ±31.77</td>
<td>1052.58 ±20.53</td>
</tr>
<tr>
<td>4</td>
<td>115.16 ±1.02</td>
<td>391.05 ±2.64</td>
<td>755.91 ±6.89</td>
<td>1013.65 ±7.81</td>
<td>1022.45 ±37.55</td>
</tr>
<tr>
<td>8</td>
<td>138.34 ±1.06</td>
<td>439.70 ±3.38</td>
<td>782.21 ±10.51</td>
<td>999.46 ±6.90</td>
<td>1045.53 ±4.19</td>
</tr>
</tbody>
</table>

GFlops/s

Time: 0.31 (s)  Gain: 0.31 (s)
PLASMA* version multiCPUs multiGPUs

- Extension to provide GPUs implementation of some internal tasks

Matrix size 8192
Matrix size 16384
PLASMA* version multiCPUs multiGPUs

- Extension to provide GPUs implementation of some internal tasks
PLASMA* version multiCPUs multiGPUs

• Extension to provide GPUs implementation of some internal tasks

- dgetrf

Matrix size 8192

Matrix size 16384
PLASMA* version multiCPUs multiGPUs

- Extension to provide GPUs implementation of some internal tasks

**dgeqrf**

Matrix size 8192

Matrix size 16384