Dealing with Heterogeneous Multicores

François Bodin

INRIA-UIUC, June 12th, 2009
Introduction

- Main stream applications will rely on new multicore / manycore architectures
  - It is about performance not parallelism

- Various heterogeneous hardware
  - General purpose cores
  - Application specific cores – GPU (HWA)

- HPC and embedded applications are increasingly sharing characteristics
Manycore Architectures

- **General purpose cores**
  - Share a main memory
  - Core ISA provides fast SIMD instructions
- **Streaming engines / DSP / FPGA**
  - Application specific architectures ("narrow band")
  - Vector/SIMD
  - Can be extremely fast
- **Hundreds of GigaOps**
  - But not easy to take advantage of
  - One platform type cannot satisfy everyone
- **Operation/Watt is the efficiency scale**
Multicore/Manycore Workload

- Multiple applications sharing the hardware
  - Multimedia, game, encryption, security, health, ...

- Unfriendly environment with many competitions
  - Global resource allocation, no warranty on availability
  - Must be taken into account when programming/compiling

- Applications cannot always be recompiled
  - Most applications are distributed as binaries

- A binary will have to run on many platforms
  - Forward scalability or “write once, run faster on new hardware”
  - Loosing performance is not an option
Multiple Parallelism Levels

- Amdahl’s law is forever, all levels of parallelism need to be exploited
  - Hybrid parallelism needed
- Programming various hardware components of a node cannot be done separately
The Past of Parallel Computing, the Future of Manycores?

- The Past
  - Scientific computing focused
  - Microprocessor or vector based, homogeneous architectures
  - Trained programmers willing to pay effort for performance
  - Fixed execution environments

- The Future
  - New applications (multimedia, medical, ...)
  - Thousands of heterogeneous systems configurations
  - Unfriendly execution environments

INRIA-UIUC, June 2009
Manycore = Numerous Configurations

- Heterogeneity brings a lot of configurations
  Proc. x Nb Cores x HWA x Mem. Sys. = 1000s of configurations

- Code optimization strategy may differ from one configuration to another

Is it possible to make a single (a few) binary that will run efficiency on a large set of configurations?
Asymmetric Behavior Issue

- Cannot assume that all cores with same ISA provide equal performance
  - Core frequency/voltage throttling can change computing speed of some cores
    - e.g. Nehalem “turbo mode”
  - Simple (in order) versus complex (out-of-order) cores
  - Data locality effects
  - ...

*How to deal with non homogeneous core behavior?*
Manycore = Multiple μ-Architectures

- Each μ-architecture requires different code generation/optimization strategies
  - Not one compiler in many cases
- High performance variance between implementations
  - ILP, GPCore/TLP, HWA
- Dramatic effect of tuning
  - Bad decisions have a strong effect on performance
  - Efficiency is very input parameter dependent
  - Data transfers for HWA add a lot of overheads

How to organize the compilation flow?
CAPS Compiler Flow for Heterogeneous Targets

- Dealing with various ISAs
- Not all code generation can be performed in the same framework
Heterogeneous Tuning Issue Example

```c
#pragma hmpp astex_codelet_1 codelet &
#pragma hmpp astex_codelet_1 , args[c].io=in &
#pragma hmpp astex_codelet_1 , args[v].io=inout &
#pragma hmpp astex_codelet_1 , args[u].io=inout &
#pragma hmpp astex_codelet_1 , target=CUDA &
#pragma hmpp astex_codelet_1 , version=1.4.0

void astex_codelet_1(float u[256][256][256], float v[256][256][256], float c[256][256][256],
                     const int K, const float x2){

    astex_thread_begin:{
        for (int it = 0 ; it < K ; ++it){
            for (int i2 = 1 ; i2 < 256 - 1 ; ++i2){
                for (int i3 = 1 ; i3 < 256 - 1 ; ++i3){
                    for (int i1 = 1 ; i1 < 256 - 1 ; ++i1){
                        float coeff = c[i3][i2][i1] * c[i3][i2][i1] * x2;
                        float sum = u[i3][i2][i1 + 1] + u[i3][i2][i1 - 1];
                        sum += u[i3][i2 + 1][i1] + u[i3][i2 - 1][i1];
                        sum += u[i3 + 1][i2][i1] + u[i3 - 1][i2][i1];
                        v[i3][i2][i1] = (2. - 6. * coeff) * u[i3][i2][i1] + coeff * sum - v[i3][i2][i1];
                    }
                }
            }
        }
    }
}

astex_thread_end;
```

Need interchange
If aims at NVIDIA GPU
Varying Available Resources

- Available hardware resources are changing over the execution time
  - Not all resources are time-shared, e.g. a HWA may not be available
  - Data affinity must be respected

How to ensure that conflicts in resource usage will not lead to global performance degradation?
OpenMP in Unfriendly Environment

- OpenMP programs performance is strongly degraded when sharing resources
  - Example with NAS parallel benchmark, 2 cores, one rogue application using one of the core
  - Best loop scheduling strategy not identical on loaded or unloaded machine
Peak Performance is Not the Goal

- Maximizing the Return on Investment
Difficult Decisions Making with Alternative Codes (Multiversioning)

- Various implementations of routines are available or can be generated for a given target
  - CUBLAS, MKL, ATLAS, ...
  - SIMD instructions, GPcore, HWA, Hybrid

- No strict performance order
  - Each implementation has a different performance profile
  - Best choice depends on platform and runtime parameters

- Decision is a complex issue
  - How to produce the decision?
Illustrating Example:
Dealing with Multiple BLAS Implementations

- Runtime selection of DGEMM in High Performance Linpack
  - Intel(R) Xeon(R) E5420 @ 2.50GHz
  - CUBLAS - Tesla C1060, Intel MKL

- Three binaries of the application
  - Static linking with CUBLAS
  - Static linking with MKL
  - Library mix with selection of routine at runtime
    - Automatically generated using CAPS tooling

- Three hardware resource configurations
  - GPU + 1, 2, and 4 cores used for MKL
Performance Using One Core

- Performance in Gigaflops
- 4 problem sizes: 64, 500, 1200, 8000

INRIA-UIUC, June 2009
Performance Using Four Cores

INRIA-UIUC, June 2009
The Challenges

- **Programming**
  - Medium

- **Resources management**
  - Medium

- **Application deployment**
  - Hard

- **Portable performance**
  - Extremely hard
Research Directions

- **New Languages**
  - X10, Fortress, Chapel, PGAS languages, OpenCL, MS Axum, ...

- **Libraries**
  - Atlas, MKL, Global Array, Spiral, Telescoping languages, TBB, ...

- **Compilers**
  - Classical compiler flow needs to be revisited
  - Acknowledge lack of static performance model
  - Adaptative code generation

- **OS**
  - Virtualization/hypervisors

- **Architectures**
  - Integration on the chip of the accelerators
    - AMD Fusion, ...
  - Alleviate data transfers costs
    - PCI Gen 3x, ...

Key for the short/mid term
Directives Based Approach for Hardware Accelerators (HWA)

- Directives
  - Do not require a new programming language
  - Already state of the art approach (e.g. OpenMP)
  - Keep incremental development possible
  - Avoid exit cost

- Does not address very large scale parallelism
  - But this is not (yet) the issue for multicore nodes

- Path chosen by CAPS entreprise
  - Heterogeneous Multicore Parallel Programming (HMPP)
  - Centered on the codelet / pure function concept
  - Focus on CPU – GPU communications optimizations
  - Complementary to OpenMP and MPI
What is Missing in OpenMP for HWA

- Remote Procedure Call (RPC) on a HWA
  - Code generation for GPU, ...
  - Hardware resource management

- Dealing with non shared address space
  - Explicit communications management to optimize the data transfers between main the CPU and the HWA
HMPP1.5 Simple Example

```c
#pragma hmpp sgemmlabel codelet, target=CUDA, args[vout].io=inout
extern void sgemm( int m, int n, int k, float alpha,
    const float vin1[n][n], const float vin2[n][n],
    float beta, float vout[n][n] );

int main(int argc, char **argv) {
    ...
    for( j = 0 ; j < 2 ; j++ ) {
#pragma hmpp sgemmlabel callsite
        sgemm( size, size, size, alpha, vin1, vin2, beta, vout );
    }
}

#pragma hmpp label codelet, target=CUDA:BROOK, args[v1].io=out
#pragma hmpp label2 codelet, target=SSE, args[v1].io=out, cond="n<800"
void MyCodelet(int n, float v1[n], float v2[n], float v3[n])
{ int i;
    for (i = 0 ; i < n ; i++) {
        v1[i] = v2[i] + v3[i];
    }
}
```

INRIA UIUC, June 2009
Group of Codelets (HMPP 2.0)

- Several callsites grouped in a sequence corresponding to a given device
  - Memory allocated for all arguments of all codelets
  - Allow for resident data but no consistency management
Optimizing Communications

- Exploit two properties
  - Communication / computation overlap
  - Temporal locality of RPC parameters

- Various techniques
  - Advancedload and Delegatedstore
  - Constant parameter
  - Resident data
  - Actual argument mapping
Advancedload Directive

- Avoid reloading constant data

```c
int main(int argc, char **argv) {
    ...
    #pragma hmpp simple advancedload, args[v2],
    for (j=0; j<n; j++) {
        #pragma hmpp simple callsite, args[v2].advancedload=true
        simplefunc1(n, t1[j], t2, t3[j], alpha);
    }
    #pragma hmpp label release
    ...
}
```

`t2` is not reloaded at each loop iteration

INRIA-UIUC, June 2009
Actual Argument Mapping

- Allocate arguments of various codelets to the same memory space
  - Allow to exploit reuses of argument to reduce communications
- Close to equivalence in Fortran

```c
#pragma hmpp <mygp> group, target=CUDA
#pragma hmpp <mygp> map, args[f1::inm; f2::inm]

#pragma hmpp <mygp> f1 codelet, args[outv].io=inout
static void matvec1(int sn, int sm,
    float inv[sn], float inm[sn][sm],
    float outv[sm])
{
    ...
}
#pragma hmpp <mygp> f2 codelet, args[v2].io=inout
static void otherfunc2(int sn, int sm,
    float v2[sn], float inm[sn][sm])
{
    ...
}
```

Arguments share the same space on the HWA.
Conclusion

- Multicore ubiquity is going to have a large impact on software industry
  - New applications but many new issues
- Will one parallel model fit all?
  - Surely not but multi languages programming should be avoided
  - Directive based programming is a safe approach
  - Ideally OpenMP will be extended to HWA
- Toward Adaptative Parallel Programming
  - Compiler alone cannot solve it
  - Compiler must interact with the runtime environment
  - Programming must help expressing global strategies / patterns
  - Compiler as provider of basic implementations
  - Offline-Online compilation has to be revisited