Programming parallel and Heterogeneous Platforms with StarSs

Rosa M. Badia
Computer Sciences Research Dept.
BSC
Barcelona Supercomputing Center

- The BSC-CNS objectives:
  - Supercomputing services and support to external research.
  - R&D in Computer Sciences, Life Sciences and Earth Sciences.

- BSC-CNS is a consortium that includes:
  - the Spanish Government (MEC) – 51%
  - the Catalanian Government (DIUE) – 37%
  - the Technical University of Catalonia (UPC) – 12%

- 300 people
BSC-CNS: Computer Sciences

Computer architecture:
- Superscalar and VLIW
- Hardware multithreading
- Design space exploration for multicore chips and Hw accelerators
- Transactional memory (Hw, Hw-assisted)
- SIMD and vector extensions/units

Benchmarking, analysis and prediction tools:
- Tracing scalability
- Pattern and structure identification
- Visualization and analysis
- Processor, memory, network, system

Programming models:
- StarSs
- OpenMP for multicore, SMP and ccNUMA
- DSM for clusters
- Transactional Memory
- Embedded architectures

Grid and cluster computing:
- Programming models (COMPSs)
- Resource management
- Distributed platforms / cloud

Operating environments:
- Autonomic application servers
- Resource management for heterogenous workloads
- Coordinated scheduling and resource management
- Parallel file system scalability
Legacy codes in PetaFlop machines

- Mostly MPI codes
- Scientific codes: engineering, molecular dynamics, geology, plasma physics ...
- Some hybrid with OpenMP
- Designed for at most thousands of processors

Current efforts
- Trying to extend to larger number of cores
- Combining with CUDA to tackle GPUs
- Developers know few about the actual behavior in the platform
Parallel programming models

- Traditional programming models
  - Message passing (MPI)
  - OpenMP
  - Hybrid MPI/OpenMP
- Heterogeneity
  - CUDA
  - OpenCL
  - ALF
  - RapidMind
- Alternative approaches
  - Partitioned Global Address Space (PGAS) programming models
    - UPC, X10, Chapel
  - ...

We need simple programming paradigms that enable easy application development
Features that can help reducing the gap:

- Sequential code closer to parallel code
- Support for heterogeneity
- Same source for different platforms
  - Code independent of underlying HW
- Tools that help on
  - Application parallelization
  - Application debug
  - Application performance analysis
Agenda

- StarSs overview
  - Data-flow execution
  - Syntax
  - Environment
- Tools
  - Taskifying applications: Ssgrind
  - Debugging
  - Performance analysis
- Example
- Conclusions
StarSs: data-flow execution of sequential programs

```c
void Cholesky( float *A ) {
    int i, j, k;
    for (k=0; k<NT; k++) {
        spotrf (A[k*NT+k]) ;
        for (i=k+1; i<NT; i++)
            strsm (A[k*NT+k], A[k*NT+i]);
        // update trailing submatrix
        for (i=k+1; i<NT; i++) {
            for (j=k+1; j<i; j++)
                sgemm( A[k*NT+i], A[k*NT+j], A[j*NT+i]);
            ssyrk (A[k*NT+i], A[i*NT+i]);
        }
    }
}
```

```c
void spotrf (float *A);
void ssyrk (float *A, float *C);
void sgemm (float *A, float *B, float *C);
void strsm (float *T, float *B);
```
void Cholesky( float *A ) {
    int i, j, k;
    for (k=0; k<NT; k++) {
        spotrf (A[k*NT+k]) ;
        for (i=k+1; i<NT; i++)
            strsm (A[k*NT+k], A[k*NT+i]);
        // update trailing submatrix
        for (i=k+1; i<NT; i++) {
            for (j=k+1; j<i; j++)
                sgemm( A[k*NT+i], A[i*NT+i], A[j*NT+i]);
            ssyrk (A[k*NT+i], A[i*NT+i]);
        }
    }
}

#pragma omp task inout ([TS][TS]A)
void spotrf (float *A);
#pragma omp task input ([TS][TS]A) inout ([TS][TS]C)
void ssyrk (float *A, float *C);
#pragma omp task input ([TS][TS]A,[TS][TS]B) inout ([TS][TS]C)
void sgemm (float *A, float *B, float *C);
#pragma omp task input ([TS][TS]T) inout ([TS][TS]B)
void strsm (float *T, float *B);

StarSs: data-flow execution of sequential programs

Decouple
how we write
form
how it is executed

Write

Execute

Rosa M. Badia, CHPC, Pretoria, December 2011
The StarSs programming model

- **StarSs**
  - A “node” level programming model
  - Sequential C/Fortran/Java + compiler directives
  - Task based
  - Simple linear address space
  - Nicely integrates with MPI
  - Natural support for heterogeneity

- **Programmability/Portability**
  - Incremental parallelization/restructure
  - Focus in the problem, not in the hardware
  - Top/down programming
  - “Same” source code runs on “any” machine
    - Optimized task implementations

- **Performance**
  - Intelligent Runtime
    - Asynchronous (data-flow) execution and locality awareness
    - Automatically extracts and exploits parallelism
    - Malleable, matches computations to specific resources on each type of target platform

Open Source
http://pm.bsc.es/ompss/
OmpSs syntax: a small set of compiler directives

```c
#pragma omp target device ({ smp | cuda })
 [ implements ( function_name )]
 { copy_deps | [ copy_in ( array_spec ,...)] [ copy_out (...)] [ copy_inout (...)] }
```

Ask the runtime to ensure data is accessible in the address space of the device

```c
#pragma omp task [ input (...)] [ output (...)] [ inout (...)] [ concurrent (...)]
 { function or code block }
```

To compute dependences

```c
#pragma omp taskwait [on (...)] [noflush]
```

Wait for sons or specific data availability

To allow concurrent execution of commutative tasks

Relax consistency to parent
Environment: Mercurium Compiler

- Recognizes constructs and transforms them to calls to the runtime
- Manages code restructuring for different target devices
  - Device-specific handlers
  - May generate code in a separate file
  - Invokes different back-end compilers
    → nvcc for NVIDIA
Environment: NANOS++ runtime structure

- Support to different programming models: OpenMP (OmpSs), StarSs, Chapel
- Independent components for thread, task, dependence management, task scheduling, ...
- Most of the runtime independent of the target architecture: SMP, GPU, tasksim simulator, cluster
- Support to heterogeneous targets
  - i.e., threads running tasks in regular cores and in GPUs
- Instrumentation
  - Generation of execution traces
Environments: OmpSs runtime behaviour

- Task management: generation, data dependence analysis, task scheduling
- Coherence support
  - A hierarchical directory
  - A software cache per GPU/cluster node
- Automatic handling of Multi-GPU execution
  - One manager thread for GPU: data transfers, task execution, synchronization
- Clusters: One runtime instance per node
  - One master image
  - N-1 worker images
  - Communication thread
  - Data transfers
Developing a StarSs application

- **Steps**
  - Develop a correct sequential application
    - ... re-structure an existent parallel one (MPI)
  - Find tasks
  - Debug application
  - Performance analysis and tuning
Developing a StarSs application

- Find tasks
  - What can be a task?
    - Piece of computation with enough granularity
    - Potential for concurrency with other tasks
    - Enabler for more concurrency or tasks generation ➞ avoid bottlenecks

Still difficult, need for some automatism
Developing a StarSs application

• Find tasks
  • What can be a task?
    • Piece of computation with enough granularity
    • Potential for concurrency with other tasks
    • Enabler for more concurrency or tasks generation ➔ avoid bottlenecks
  • Further exploitation of finer parallelism through nesting
    • Good paradigm to structure codes top down
    • Support for hierarchy/heterogeneity
Ssgrind: Finding tasks

Valgrind based instrumentation
- Track task entry and exit
- Track mallocs
- Track ALL memory accesses → dependences
- Estimate time = f(#instructions)

Sequentia
l code

Code translati
on

cc

Executio
n

Valgrind
tracer

Dimemas
simulator

Original execution

Potential StarSs execution

Incomplete/suggested taskification
- Compiler directives
- Runtime calls (even on not well structured code)

“Quantifying the potential task-based dataflow parallelism in MPI applications”. V. Subotic et al. Europar 2011
Ssgrind: replaying with Dimemas

- Trace generation
  - Includes original application information (MPI)
  - Tasks and dependences information

**MPI execution**

- MPI code
- Code translation
- mpicc
- ... MPI process...
- Valgrind tracer
- Trace expander
- Dimemas simulator
  - Original MPI execution
  - Potential MPI/SMPSs' execution

- Configurable architecture description
  - Change of CPU speed
  - Change of speed of a specific task
  - Trace replay with Dimemas
Possible taskifications

The parallelism is released in transition T3 -> T4 when function `update` is separated from the rest of the inner loop.
Debugging

- Methodology first Step: Serial
  - 1 core in order, 1 core out of order
  - Specify phases in source

- Conforming to pragmas?

- Task based debugging
  - Visualize graph
  - Task based breakpointing
  - Control scheduling

- Interaction to classical sequential debugger

“StarSscheck: A tool to Find Errors in Task-Based Parallel Programs ”. P. M. Carpenter et al. Europar 2010
Performance analysis with CEPBA-Tools

- Performance analysis and simulation toolset
  - Tracing
  - Simulation
  - Visualization and analysis
    - Large number of options: different views, statistics

- Detailed analysis enables fine tuning of codes
int main (int argc, char **argv) {
    int i, j, k;
    ...
    initialize(A, B, C);

    for (i=0; i < NB; i++)
        for (j=0; j < NB; j++)
            for (k=0; k < NB; k++)
                mm_tile( C[i][j], A[i][k], B[k][j]);
}

#pragma omp task input([BS][BS]A, [BS][BS]B)\inout([BS][BS]C)
static void mm_tile ( float C[BS][BS], float A[BS][BS],
                        float B[BS][BS]) {
    int i, j, k;

    for (i=0; i < BS; i++)
        for (j=0; j < BS; j++)
            for (k=0; k < BS; k++)
                C[i][j] += A[i][k] * B[k][j];
}

Will work on matrices of any size

Will work on any number of cores/devices
int main (int argc, char **argv) {
    int i, j, k;

    initialize(A, B, C);

    for (i=0; i < NB; i++)
        for (j=0; j < NB; j++)
            for (k=0; k < NB; k++)
                mm_tile( C[i][j], A[i][k], B[k][j], BS);
}

#pragma omp target device (cuda) copy_deps
#pragma omp task input([NB][NB]A, [NB][NB]B, NB) \  
inout([NB][NB]C)
void mm_tile (float *A, float *B, float *C, int NB) {
    unsigned char TR = 'T', NT = 'N';
    float DONE = 1.0, DMONE = -1.0;
    float *d_A, *d_B, *d_C;

    cublasSgemm (NT, NT, NB, NB, NB, DMONE, A, 
                 NB, B, NB, DONE, C, NB);
}
int main (int argc, char **argv) {
    int i, j, k;
    ...

    initialize(A, B, C);

    for (i=0; i < NB; i++)
        for (j=0; j < NB; j++)
            for (k=0; k < NB; k++)
                mm_tile( C[i][j],

#pragma omp target device (cuda)
#pragma omp task input([NB][NB]A, [NB][NB]B, NB) inout([NB][NB]C)
void mm_tile (float *A, float *B, float *C) {
    int hA, wA, wB;
    hA = NB; wA = NB; wB = NB;

    dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);
    dimBlock.x = BLOCK_SIZE;
    dimBlock.y = BLOCK_SIZE;
    dim3 dimGrid;
    dimGrid.x = (wB / dimBlock.x);
    dimGrid.y = (hA / dimBlock.y);
    Muld <<<dimGrid, dimBlock>>> ( A, B, wA, wB, C );

    __global__ void Muld(float* A, float* B, int wA, int wB, float* C) {
        int bx = blockIdx.x; int by = blockIdx.y;
        int tx = threadIdx.x; int ty = threadIdx.y;
        int aBegin = wA * BLOCK_SIZE * by;
        int aEnd = aBegin + wA - 1;
        int aStep = BLOCK_SIZE;
        int bBegin = BLOCK_SIZE * bx;
        int bStep = BLOCK_SIZE * wB;
        float Csub = 0;

        for (int a = aBegin, b = bBegin; a <= aEnd; a += aStep, b += bStep) {
            __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
            __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];
            As[ty][tx] = A[a + wA * ty + tx];
            Bs[ty][tx] = B[b + wB * ty + tx];
            __syncthreads();

            for (int k = 0; k < BLOCK_SIZE; ++k)
                Csub += As[ty][k] * Bs[k][tx];
            __syncthreads();
        }

        int c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx;
        C[c + wB * ty + tx] += Csub;
    }
}
Matrix multiply: multi-GPUs

Two Intel Xeon E5440, 4 cores
4 Tesla S2050 GPUs

- Main thread
- GPU thread
- Copying data to GPU
- All copies to host at end of application
Matrix multiply: clusters of GPUs

2 nodes of DAS-4
each node:
Two Intel Xeon E5620, 4 cores
1 GTX 480 GPU
QDR Infiniband
Cholesky performance

- Matrix size: 16K x 16K
- Block size: 2K x 2K
- Storage: Blocked / linear
- Overlap CPU/GPU tasks
- Tasks:
  - spotrf: Magma
  - trsm, syrk, gemm: CUBLAS

Double precision

Two Intel Xeon E5440, 4 cores
4 Tesla S2050 GPUs
A quiet revolution

- A change in mentality

  Bottom up and being in total control
  Fork join, data parallel, explicit data placement

  Top down, potentials and hints rather than exact howtos
  Asynchrony, data flow, automatic locality management

- Deeply rooted (in or genes), but need to overcome our fears.
  - May require some effort, but it is possible and there is a lot to gain
  - Understanding and confidence through tools will be key
  - Need education from very early levels (shape and not reshape minds)

- Adaptability/Flexibility is key to survive in rapidly changing environments

- We believe (MPI/)StarSs is a sensible approach to exascale and before
Availability and information

- Software availability
  www.project-text.eu
  pm.bsc.es
The TEXT project

- Towards EXaflop applicaTions

- Demonstrate that Hybrid MPI/SMPSs addresses the Exascale challenges in a productive and efficient way.
  - Deploy at supercomputing centers
  - Port Applications (HLA, SPECFEM3D, PEPC, PSC, BEST, CPMD, LS1 MarDyn) and develop algorithms.
  - Develop additional environment capabilities
    - tools (debug, performance)
    - improvements in runtime systems (load balance and GPUss)
  - Support other users
    - Identify users of TEXT applications
    - Identify and support interested application developers
  - Contribute to Standards (OpenMP ARB, PERI-XML)
Thank you