Extending the scope and support for StarSs

Jesus Labarta
Director Computer Sciences Research Dept.
BSC
Current architectures → ISA leaks

Opportunity, business model for “macho” programmers
Globally not so good

Control flows
Memory architecture
hierarchy
explicit transfers

Dazzled by performance
Need to reintroduce “seny”: a granularity issue

High level application structure should not change with changing target architecture

A clean “abstract” interface
Tasks: high flexibility “user defined ISA”
Potentially multiple implementations
target of specific optimizations

Power to the Runtime
Self similar hierarchical architecture
“Reuse”/architectural ideas extend under new constraints
The StarSs family of models: key concept

- Sequential program …
  - Task based program on single address/name space
    - Blocked to achieve sufficient task granularity → performance
    - Discipline
    - Order IS defined !!!!
  - Directionality annotations
    - Used to **compute dependences** AND to provide **data access information**
    - Use pattern, NOT resources and forcing actions (copies,…)

- … happens to execute in parallel
  - Automatic run time computation and honoring of dependencies
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++)
            strm (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]);
            ssysr (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 ssysr (float *A, float *C);
#pragma omp task input ([TS][TS]A,[TS][TS]B) inout ([TS][TS]C )
 void ssgem (float *A, float *B, float *C);
#pragma omp task input ([TS][TS]T) inout ([TS][TS]B)
 void strsm (float *T, float *B);
StarSs: Enabler for exascale

- Can exploit very unstructured parallelism
  - Not just loop/data parallelism
  - Easy to change structure
- Supports large amounts of lookahead
  - Not stalling for dependence satisfaction
- Allow for locality optimizations to tolerate latency
  - Overlap data transfers, prefetch
  - Reuse
- Nicely hybridizes into MPI/StarSs
  - Propagates to large scale the node level dataflow characteristics
  - Overlap communication and computation
  - A chance against Amdahl’s law

- Support for heterogeneity
  - Any # and combination of CPUs, GPUs
  - Including autotuning
- Malleability: Decouple program from resources
  - Allowing dynamic resource allocation and load balance
  - Tolerate noise

Data-flow; Asynchrony

Potential is there; Can blame runtime

Compatible with proprietary low level technologies
StarSs: history/strategy/versions

Basic SMPs:
- must provide directionality ∀ argument
- Contiguous, non partially overlapped
- Renaming
- Several schedulers (priority, locality, …)
- No nesting
- C/Fortran
- MPI/SMPs optimisations

SMPSs regions:
- C, No Fortran
- must provide directionality ∀ argument
- overlapping & strided
- Reshaping strided accesses
- Priority and locality aware scheduling

OMPSs:
- C/C++, Fortran under development
- OpenMP compatibility (~)
- Contiguous args. (address used as centinels)
- Separate dependences/transfers
- Inlined/outlined pragmas
- Nesting
- Heterogeneity: SMP/GPU/Cluster
- No renaming,
- Several schedulers: “Simple” locality aware sched,…

Evolving research since 2005
OmpSs: Directives

Task implementation for a GPU device
The compiler parses CUDA kernel invocation syntax

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

Support for multiple implementations of a task

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

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

To compute dependences

To allow concurrent execution of commutative tasks

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

Wait for sons or specific data availability

Relax consistency to main program
All that easy/wonderful?

• Difficulties for adoption
  • Chicken and egg issue users ↔ manufacturers
  • Availability. Development as we go
    • To all models, by all developers (Companies, research,…)
    • Willingness to contribute: Not done here, …

• Lack of program development support
  • Understand application dependences
  • Understand potential and best direction

• Difficulties of the models themselves
  • Simple concepts take time to be matured
  • As clean/elegant as we claim?
  • Legacy sequential code less structured than ideal

Early adopters and porting

Research support:
  • Consolider (Spain)
  • TEXT, Montblanc, DEEP (EC)

Standardization:
  • OpenMP, …
  • Maturity

New tools
  • Taskification
  • Performance prediction
  • Debugging

New Platforms
  • ARM + GPUs
  • MIC
  • …

Examples
Training
Education
Performance analysis
Performance analysis of dynamic task based systems

• Dynamic, variability, ... a new situation

• Profiles
  • Scalasca @ SMPSs, OmpSs
  • Metrics, first order moments ??

• Traces
  • Analysis of snapshots
  • Paraver instrumentation in all our developments
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 all. Europar 2010
Predictive environment
Different situations

• Parallelizing a “legacy” code
  • Not knowing structure, data flow, …
  • Understand potential performance
  • Large overhead acceptable
  • Suggest pragmas?

Application structure and potential can be very different from our preconceived models

Tools needed to avoid flying blind in the midst

• Understanding an MPI/SMPSs
  • Large programs
  • Where to focus improvement of tiles
  • How will it scale? How will bottlenecks shift?
Environment: Analyzing “legacy” codes

“Quantifying the potential task-based dataflow parallelism in MPI applications”. V. Subotic et al. Europar 2011
Environment: Input

Dimemas:
- Distributed machine simulator
- Trace of MPI run
- Trace of MPI/SMPSs run

Original (MPI) execution

Potential (MPI+SMPSs) execution

Incomplete/suggested taskification
- pragmas
- runtime calls (even on not well structured code)
Environment: code translation

input code

```c
#pragma css task
void compute(float *A, float *B) {
...
}
int main () {
...
compute(a,b);
...
start_task_valgrind(“section2”);
compute2(a,b);
compute3(a,b);
end_task_valgrind();
...
}
```

translated code

```c
void compute(float *A, float *B) {
...
}
int main () {
...
start_task_valgrind(“compute”);
compute(a,b);
end_task_valgrind();
...
start_task_valgrind(“section2”);
compute2(a,b);
compute3(a,b);
end_task_valgrind();
...
}```
Tracing

Input code ➔ code translation ➔ mpicc ➔ MPI execution ➔ Valgrind tracer ➔ Dimemas: Distributed machine simulator ➔ Potential (MPI+SMPSs) execution

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

Traces
- Sequential order within MPI process
- Tasks and dependences mapped to communications within process
**Possible taskifications**

```c
void update(...) {
    HPL_dtrsm(...);
    HPL_dgemm(...);
}

main() {
...
for (j = 0; j < N; j += BS)
{
    panel_init(...);
    if (cond0)
        fact(...);
    init_for_pivoting(...);
    for (i = k; i < P; i += BS)
    {
        if (cond1)
            HPL_dlaswp01N(...);
        HPL_spreadN(...);
        if (cond2)
            HPL_dlaswp06N(...);
        if (cond3)
            HPL_dlacpy(...);
        HPL_rollN(...);
        HPL_dlaswp00N(...);
        update(...);
    }
...
}
```

*HP Linpack:*
- 4 MPI processes;
- Problem size: 8192;
- Block size: 256, 128, 64, 32.

The parallelism is released in transition T3 -> T4 when function `update` is separated from the rest of the inner loop.
Taskification T2

*HP Linpack:*
- 4 MPI processes;
- Problem size: 8192;
- Block size: 256

Version T2

Target platform:
- infinite number of cores per node
Taskification T4

*HP Linpack:
  4 MPI processes;
  Problem size: 8192;
  Block size: 256
Version T4
Target platform:
infinite number of cores per node
Comparison
Still not perfect … but can analyze
Parallelism and Parallel Efficiency

Increased parallelism gives no performance benefit on a small parallel machine.

Increased parallelism gives better efficiency on a big parallel machine.
Directionality analysis

- Three levels of output
- Individual task
- Per task summary
- Directionality hints

```
task name: compute, total number of instances 40
  parameter no 1: A
    UNION OF INPUTS: [0, 1600)
    UNION OF OUTPUTS:
  parameter no 2: B
    UNION OF INPUTS:
    UNION OF OUTPUTS: [0, 1600)
  parameter no 3: C
    UNION OF INPUTS: [0, 1600)
    UNION OF OUTPUTS: [0, 1600)

task name: compute_strided, total number of instances 40
  parameter no 1 A
    UNION OF INPUTS: [0, 160), [320, 480), [640, 800), [960, 1120), [1280, 1440)
    UNION OF OUTPUTS:
  parameter no 2: B
    UNION OF INPUTS:
    UNION OF OUTPUTS: [0, 160), [320, 480), [640, 800), [960, 1120), [1280, 1440)
  parameter no 3: C
    UNION OF INPUTS: [0, 160), [320, 480), [640, 800), [960, 1120), [1280, 1440)
    UNION OF OUTPUTS: [0, 160), [320, 480), [640, 800), [960, 1120), [1280, 1440)
```

compute:
  input(A_in[0,1600]) output(B_out[0,1600]) inout(C_inout[0,1600])

compute_strided:
  input(A_in[0,1440]) output(B_out[0,1440]) inout(C_inout[0,1440])
Environment: Analyzing MPI/SMPSs applications

Traces
- Sequential order within MPI process
- Tasks and dependences mapped to communications within process

Compiler injected instrumentation
- MPI calls
- Track task entry and exit
- Measured time
Where to devote effort?

Profile of single process app

<table>
<thead>
<tr>
<th>task name</th>
<th>percentage [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>main_task</td>
<td>0.28</td>
</tr>
<tr>
<td>bdiv</td>
<td>2.76</td>
</tr>
<tr>
<td>block_mpy_add</td>
<td>54.44</td>
</tr>
<tr>
<td>bmod</td>
<td>38.29</td>
</tr>
<tr>
<td>fwd</td>
<td>3.49</td>
</tr>
<tr>
<td>lu0</td>
<td>0.53</td>
</tr>
</tbody>
</table>

What if: 2x in one routine
Decision depends on core count

Detail is important
Scheduling is important
Where to devote effort? Impact?

Profile of MPI application

<table>
<thead>
<tr>
<th>task name</th>
<th>percentage [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>fact</td>
<td>0.49</td>
</tr>
<tr>
<td>dlaswp00N</td>
<td>0.34</td>
</tr>
<tr>
<td>dlaswp01N</td>
<td>0.63</td>
</tr>
<tr>
<td>dlaswp06N</td>
<td>1.29</td>
</tr>
<tr>
<td>update</td>
<td>96.99</td>
</tr>
</tbody>
</table>

What if: 2x in one routine
Decision depends on core count

Combined architecture and application

⇒ Dynamic, Adaptive
⇒ Intelligence to runtime
To fine grain architectural simulations ...

- Tasksim:
  - Integrated in OmpSs: traces and runtime
- Example studies:
  - Tolerance to latency, need bandwidth
  - Use a shared last-level cache to filter bandwidth (not for latency)

---


• PRACE prototype @ BSC: ArM + mobile GPU

Tegra3 Q7 module:
1x Tegra3 SoC
4x Corext-A9 @ 1.5 GHz
4 GB DDR3 DRAM
6 GFLOPS
~4 Watt
1 Gbe interconnect

Nvidia GeForce 520MX
48 CUDA cores @ 900 MHz
142 GFLOPS
12 Watts
11.8 GFLOPS / W

1U Multi-board container:
1x Board container
8x Q7 carrier boards
32x ARM Corext-A9 Cores
8x GT520MX GPU
32 GB DDR3 DRAM
1.2 TFLOPS
~140 Watt

Rack:
32x Board container
10x 48-port 1GbE switches
256x Q7 carrier boards
256x Tegra3 SoC
1024x ARM Corext-A9 Cores
256x GT520MX GPU
1TB DDR3 DRAM
38 TFLOPS
~5 Kwatt
7.5 GFLOPS / W

• Mont-blanc (ICT-288777)
  • EC funded project, Just started
  • Low power, Embedded technology → exascale
  • Low power components + MPI/OmpSs
  • Integrated prototype, applications and future system design
Applications
Undertaken significant application porting efforts

- TEXT EC project:
  - Scalapack, PLASMA, SPECFEM3D, LBC, CPMD PSC, PEPC, LS1 Mardyn, Asynchronous algorithms, Microbenchmarks
- Montblanc EC project
  - YALES2, EUTERPE, SPECFEM3D, MP2C, BigDFT, QuantumESPRESSO, PEPC, SMMP, ProFASI, COSMO, BQCD
- DEEP EC project

- Consolider project (Spanish ministry)
  - MRGENESIS
- BSC initiatives and collaborations:
  - GROMACS, GADGET, WRF,…
Codes being ported

- Scalapack: Cholesky factorization (UJI)
  - Example of the issues in porting legacy code
  - Demonstration that it is feasible
  - The importance of scheduling

- LBC Boltzmann Equation Solver Tool (HLRS)
  - Solver for incompressible flows based on Lattice-Boltzmann methods (LBM)
  - Stencil and sub domains → leverage Fortran pointers to subarrays
Codes being ported

- **PEPC (JSC)**
  - N-body Coulomb. ‘Hashed-oct-tree’ algorithm using multipole expansions
  - Application restructuring
  - New functionalities in SMPSs that may be addressed by nesting in OmpSs

- **SPECFEM3D (UPPA)**
  - Simulation of 3D seismic waves propagation
  - Finite element code: large elements (5x5x5)
  - Simple code → overlap comm/comp

```c
for iter
Compute(red)
exchange(red, neighbors_list)
Compute (green)
#pragma css barrier
Update (red and green)
```
NOT only «scientific computing»

- Plagiarism detection
  - histograms, sorting, …

- File processing

- Clustering algorithms
  - G-means

- Image processing
  - Tracking

- Paraver
Conclusion
A quiet revolution

- A change in mentality
  - 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

- Top down, potentials and hints rather than how-tos,
  - Asynchrony, data flow, automatic locality management

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

- We believe (MPI/StarSs) is a sensible approach to exascale and before
Want to join?