OmpSs
A proposal for programming heterogeneous multi-core architectures

Authors: Alejandro Duran, Eduard Ayguade, Rosa M. Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, Judith Planas

Trond Inge Lillesand
TDT24
November 2012
Introduction

- Addresses complexity of code (parallel programming)
  - Makes programmers less productive
  - Decreases portability
- This is combined with an increase in programming models
  - Have to use separate models when programming SMP, heterogeneous architectures and clusters
- Proposes OmpSs
  - A solution to ease the programming of heterogeneous architectures
- Developed to run on SMP with / without GPUs. - And cluster environments.
  - Unify SMP, heterogeneous and cluster programming in one model
- OpenMP achieves high expressiveness using tasks.
- StarSs allow runtime analysis between tasks, and automatic data transfers
What is OmpSs?

- Directive-based
- Can also incorporate the use of OpenCL and CUDA kernels
- Different execution model than OpenMP (thread-pool instead of fork-join)
OmpSs : Goal

Enable OmpSs programming model to:

- Efficiently run on clusters of SMPs, including GPUs
- Develop optimizations to minimize data movement
Extensions

- OmpSs is extended with a set of constructs to specify data dependencies and/or to specify heterogeneous devices

- Data dependencies
  - input: the task will not be eligible to run as long as a previously created task with an output clause applying to the same lvalue has not finished its execution
  - output: the task will not be eligible to run as long as a previously created task with an input or output clause applying to the same lvalue has not finished its execution
  - inout: the task that evaluates to a given lvalue is considered to have an input clause and an output clause that evaluates to that same lvalue
  - inout-set: same as inout, except that it will not create dependencies to other tasks with an inout-set clause evaluating to the same lvalue
Example

void foo ( int *a, int *b )
{
    for ( i = 1; i < N; i++ ){
        #pragma omp task input(a[i-1]) inout(a[i]) output(b[i])
        propagate(&a[i-1],&a[i],&b[i]);

        #pragma omp task input(b[i-1]) inout(b[i])
        correct(&b[i-1],&b[i]);
    }
}
Heterogeneous Extensions

#pragma omp target [clauses]

- The target construct specifies that a given element can be run in a device
- The clauses are:
  - device: Specify target device
  - copy_in: Specifies a set of shared data that needs to be transferred to the device before the associated code can be executed
  - copy_out: Specifies a set of shared data that needs to be transferred from the device
  - copy_inout: A combination of copy_in and copy_out
  - copy_deps: Specifies that if the attached construct has any dependence clause, then they will also have copy semantics
  - implements: Specifies that the code is an alternate implementation for the target devices of the function name specified in the clause
Example 2

#pragma omp target device (cuda) copy_deeps implements (matmul_tile)
#pragma omp task inout([NB*NB]C) input([NB*NB]A, [NB*NB]B)
void matmul_tile_cuda (REAL *A, REAL *B, REAL *C)
{
    //Cuda Kernel
}

Application Kernels

- Matrix Multiplication
- Black Scholes
- Perlin Noise
- Julia Set
- Fixed-Grid
- Parallel Bayesian Phylogenetic Inference
Execution Environments

Intel Xeon Server

- 4 Intel Xeon chips (6 cores each at 2.4 GHz)
- Shared L2 cache of 12 MB
- 48 GB of Main Memory

In this SMP environment, the application kernels with OmpSs and OpenCL are tested

OmpSs avoids data copying, while OpenCL perform data copying
Execution Environments

NVIDIA GPUs

- Second testbench is hosted with dual chip AMD Opterons and 8 GB of Memory
  - Two NVIDIA GTX 285
  - Using AMD/ATI OpenCL SDK and compare to the OmpSs implementation

- A third testbench has 4 i7s with 24 GB of Main Memory
  - NVIDIA GTX 480 GPU
  - Using NVIDIA OpenCL SDK and compare to the OmpSs implementation
Results, Matrix Multiplication

Fig. 8. Evaluation of Matrix Multiply (512x512) in an Intel Xeon server

Fig. 9. Evaluation of Matrix Multiply (1024x1024) in an Intel Xeon server
Results: Matrix Multiply, Black Scholes

Fig. 10. Evaluation of Matrix Multiply (2048x2048) in an Intel Xeon server

Fig. 11. Evaluation of BlackScholes in an Intel Xeon server
Results: Perlin Noise and Julia Set

In Figure 12, the OpenCL kernel is compiled at runtime, while the green column is an example of code that can be reused.
Results: PBPI and Fixed Grid

Due to the ability of OmpSs to manage dependent tasks and reduce the number of barriers inside the application, the performance and scalability with OmpSs outperforms OpenMP. In PBPI, OmpSs performs worse on a small number of CPUs due to task creation overhead.

Fig. 14. Evaluation of PBPI in an Intel Xeon server

Fig. 15. Evaluation of FixedGrid in an Intel Xeon server
Results: Matmul And Black Scholes (GPU)

- Figure 16: Evaluation of Matrix Multiply in NVIDIA GTX285 GPUs
- Figure 17: Evaluation of BlackScholes in NVIDIA GTX285 GPUs

- Figure 16: Matrix multiplication from the CUDA and OpenCL SDK compared to the OmpSs approach with a CUDA kernel
Results : Perlin Noise and Julia Set

Fig. 18. Evaluation of Perlin Noise in NVIDIA GTX285 and GTX480 GPUs

Fig. 19. Evaluation of Julia Set in NVIDIA GTX285 and GTX480 GPUs
	nthe serial time, a slight speed-up is observed when increasing the number of GPUs (around 1.5).
Conclusion

- The paper presents OmpSs, a proposal to improve programming on multicore processors and GPUs.
- OmpSs is based on OpenMP and StarSs, taking features such as dependence analysis and automatic generation of data transfers from StarSs, and features related to tasks from OpenMP.
- The proposal improves productivity and achieves performance similar or better than CUDA/OpenCL and OpenMP on different architectures in this article.