OmpSs: Programming Clusters of GPUs Made Easy

Jesus Labarta
Director Computer Sciences Research Dept.
BSC
The StarSs family of programming models

- A “node” level programming model
- C/Fortran/Java
- Task based. Asynchrony, data-flow.
- Simple linear address space
- Malleable
- Nicely hybridizes (MPI/StarSs)
- Natural support for heterogeneity

Key concepts:
- Sequential program
- Directionality annotations on tasks arguments

Key objectives:
- Same program any platform
- Productive / Incremental programming
StarSs: data-flow execution of sequential programs

• A sequential program → order IS defined
• That happens to execute in parallel

```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[j*NT+i], A[k*NT+j]);
            ssyrk (A[k*NT+i], A[i*NT+i]);
        }
    }
}
```

```c
#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: the potential of data access information

- Flat global address space seen by programmer
- Flexibility to dynamically traverse dataflow graph “optimizing”
  - Concurrency. Critical path
  - Memory access: data transfers performed by run time

- Opportunities for
  - Prefetch
  - Reuse
  - Eliminate antidependences (rename)
  - Replication management
    - Coherency/consistency handled by the runtime
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
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

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

To compute dependences

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

Wait for sons or specific data availability

To allow concurrent execution of commutative tasks

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

Relax consistency to main program

---

Jesus Labarta, OmpSs@cluster of GPUs, NVIDIA Booth, SC 2011
void blocked_cholesky( int NT, 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]);
        }
    }
}

#pragma css task input(A[NB*NB], B[NB*NB], NB) inout(C[NB*NB]) target device (cuda)
void sgemm(float *A, float *B, float *C, unsigned long NB) {
    unsigned char TR='T', NT='N';
    float DONE=1.0, DMONE=-1.0;
    cublasSgemm( NT, TR, NB, NB, NB, DMONE, A, NB, B, NB, DONE, C, NB );
}

n = 8192; bs = 1024
Heterogeneous execution

- Available spotrf more efficient at CPU
- Overlap between CPU and GPU

```c
#pragma css task input(NB) inout(A[NB*NB]) target device (smp)
void spotrf_tile(float *A, int NB)
{
    long INFO;
    char L = 'L';

    spotrf_( &L, &NB, A, &NB, &INFO );
}
```
Perlin noise

```c
#pragma omp target device(cuda)
__global__ void cuda_perlin (pixel output [], float time, int j, int rowstride)
{
    unsigned int i = blockIdx.x * blockDim.x + threadIdx.x;
    unsigned int off = blockIdx.y * blockDim.y + threadIdx.y;
    float vdx = 0.03125f;
    float vdy = 0.0125f;
    float vs = 2.0f;
    float bias = 0.35f;
    float red, green, blue;
    float xx, yy;
    float vx, vy, vt;
    vx = ((float) i) * vdx;
    vy = ((float) j+offset) * vdy;
    vt = time * vs;
    xx = vx * vs;
    yy = vy * vs;
    red = noise3(xx, vt, yy);
    green = noise3(vt, yy, xx);
    blue = noise3(yy, xx, vt);
    red += bias;
    green += bias;
    blue += bias;
    // Clamp to within [0 .. 1]
    red = (red > 1.0f) ? 1.0f : red;
    green = (green > 1.0f) ? 1.0f : green;
    blue = (blue > 1.0f) ? 1.0f : blue;
    red = (red < 0.0f) ? 0.0f : red;
    green = (green < 0.0f) ? 0.0f : green;
    blue = (blue < 0.0f) ? 0.0f : blue;
    red *= 255.0f;
    green *= 255.0f;
    blue *= 255.0f;
    output[(off * rowstride) + i].r = (unsigned char) red;
    output[(off * rowstride) + i].g = (unsigned char) green;
    output[(off * rowstride) + i].b = (unsigned char) blue;
    output[(off * rowstride) + i].a = (unsigned char) 255;
}
```

```c
for (j = 0; j < img_height; j+=BS) {
    // BS image rows per task
    pixel *out = &output[j*rowstride];
    #pragma omp target device(cuda) copy_deps
    #pragma omp task output([rowstride*BS]out)
    {
        dim3 dimBlock;
        dim3 dimGrid;
        // dimBlock.x = (img_width < BSx) ? img_width : BSx;
        dimBlock.x = (img_width < BSx) ? img_width : BSx;
        // dimBlock.y = (BS < BSy) ? BS : BSy;
        dimBlock.y = (BS < BSy) ? BS : BSy;
        dimBlock.z = 1;
        // dimGrid.x = img_width/dimBlock.x;
        dimGrid.x = img_width/dimBlock.x;
        // dimGrid.y = BS/dimBlock.y;
        dimGrid.y = BS/dimBlock.y;
        dimGrid.z = 1;
        cuda_perlin <<<dimGrid, dimBlock>>> (out, time, j, rowstride);
    }
} #pragma omp taskwait noflush
```
void Calc_forces(Particle* particle, Particle *output, int nb_part, float t_interval) {
    const int bs = nb_particles/8;
    size_t num_threads, num_blocks;
    int first_local, last_local;
    ...
    Particle *block7 = &particle[6*bs];
    Particle *block8 = &particle[8*bs];
    for (i = 0; i < nb_particles; i += bs) {
        ...
        output_block = &output[first_local];

        #pragma omp target device(cuda) copy_in([nb_particles] particle) \
            copy_out([bs] output_block)
        #pragma omp task output([bs] output_block) \
            input([bs] block1, [bs] block2, [bs] block3, [bs] block4, \
        {
            cal_forces_kernel<<<num_blocks, num_threads>>>(t_interval, particle, nb_particles, output_block, first_local, last_local);
        }
    }
    for (timestep = 1; timestep <= nb_timesteps; timestep++) {
        Calc_forces(particles, particles_2, nb_particles, time_interval);

        Particle * tmp = particles; /* swap arrays */
        particles = particles_2;
        particles_2 = tmp;
    } /* for timestep */

    #pragma omp taskwait
Difference between …

- GPU and cluster?
- GPU and cluster with GPUs?
- Hierarchical heterogeneous architectures

The same

Separate address space
Same task offloading can be applied
Runtime Environment: NANOS++

- 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 per GPU: data transfers, task execution, synchronization
- Clusters: One runtime instance per node
  - One master image
  - N-1 worker images
  - Communication thread
  - Data transfers
nbody

1GPU

2 GPUs

1 Node

2 Nodes

4 Nodes
Jesus Labarta, OmpSs@cluster of GPUs, NVIDIA Booth, SC 2011

**Same source different targets**

### Multi-GPUs

- **MXM**
- **Perlin Noise**
- **Blackscholes**

### Clusters of GPUs

- **Two Intel Xeon E5440, 4 cores + 4 Tesla S2050 GPUs**
- **8 nodes of DAS-4 each:** Two Intel Xeon E5620, 4 cores + 1 GTX 480 GPU QDR Infiniband
Performance and internal policies

**nbody**

- Bar charts showing performance (KParrs/s) for different numbers of nodes (1, 2, 4, 8)
  - MPI+Cuda
  - OmpSs cluster

**Perlin**

- Bar charts showing performance (MPix/s) for different numbers of nodes (1, 2, 4, 8)
  - Flush MPI+Cuda
  - Flush OmpSs cluster
  - NoFlush MPI+Cuda
  - NoFlush OmpSs cluster

---

“Productive Programming of GPU Clusters with OmpSs”. V. Subotic et al. Submitted
Performance and internal policies

Matrix multiply

Stream
Fighting Amdahl’s law: StarSs. a chance for lazy programmers

Four loops/routines
Sequential program order

OpenMP
not parallelizing one loop

SMPSs
not parallelizing one loop

GROMACS
StarSs NOT only «scientific computing»

- Plagiarism detection
  - Histograms, sorting, …

- File processing

- Clustering algorithms
  - G-means

- Image processing
  - Tracking
Conclusion

• StarSs / OmpSs

  • Single program → any target
    • Homogeneous / Heterogeneous multicores
    • Single node / Cluster
    • Nicely propagates asynchrony and overlap to Hybrid
  • Leveraging CUDA kernels/libraries and GPUs performance
  • Asynchrony/dataflow execution. Huge lookahead. Malleability
  • Data access awareness
    • Minimize transfers, prefetch, …

• Homogenized view at heterogeneity

• Still a lot of room to improve the scheduling intelligence, but very promising
Cholesky performance

- Matrix size: 16K x 16K
- Block size: 2K x 2K
- Storage: Blocked / contiguous
- Tasks:
  - spotrf: Magma
  - trsm, syrk, gemm: CUBLAS

![Graph showing Cholesky performance](image)
### Matmul (16K, float, BS=1K)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>1 Node</th>
<th>2 Nodes</th>
<th>4 Nodes</th>
<th>8 Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 GPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 GPUs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Jesus Labarta, OmpSs@cluster of GPUs, NVIDIA Booth, SC 2011
Heterogeneity and StarSs

- A really heterogeneous system may have several hosts, and different types of accelerators or specific resources
- Different implementations
  - Default: every task should at least be runnable on the host
  - implementation for each specific accelerators (even alternative impls.)

```c
#pragma css target device (cell) copyin ([TS][TS]T, [TS][TS]B) copyout ([TS][TS]B)
#pragma css task input ([TS][TS]T) inout ([TS][TS]B)
void chol_strsm (float *T, float *B);

#pragma css target device (cell) copyin ([TS][TS]A, [TS][TS]C) \ 
  copyout ([TS][TS]C)
#pragma css task input ([TS][TS]A) inout ([TS][TS]C)
void chol_ssyrk (float *A, float *C);

#pragma css target device (cuda) copyin ([TS][TS]T, [TS][TS]B, [TS][TS]C) \ 
  copyout ([TS][TS]B)
#pragma css task input ([TS][TS]A, [TS][TS]B) inout ([TS][TS]C)
void chol_sgemm (float *A, float *B, float *C);
```
The potential of nesting

- Hierarchical parallelization
  - A task could become the environment where a new task graph is instantiated
  - Potential for collapsing levels

- Top down IS also good for performance
  - Exploit unstructured (heterogeneous) parallelism
  - Parallelize overheads (task generation)
• Prototipo Tegra.
  • Mucha gente les ha pedido uno.
  • Poner slides
• Cluster Bull
  • Slide
  • Decir que se anunciara?
• Mi opinion.
  • A GPU: A very fast device for one purely data parallel task
  • Advice: Forget about GPU
  • Relax: can rely on raw horse power is there.
• Why GPUs
  • Dazzling performance
  • Research center have to explore
• My BIG concern:
  • Programmers QoL: Quality of Life
  • Productivity and performance
• StarSs:
  • One program, any target. Program at an abstract algorithmic level. Possibilities rather than how tos
  • And still possible to get better performance than user level scheduled
    • Asynchronous medium grain dataflow
    • Hybridize (MPI/OmpSs9 → very large scale.
    • Prefetch, overlap communication and data transfer, Locality aware scheduling.
• Message. Forget your fears. It is possible. It will be here.
nbody

1GPU

2 GPUs

1 Node

2 Nodes

4 Nodes

8 Nodes
Perlin noise

1 Node

2 Nodes

4 Nodes

8 Nodes