From the latency to the throughput age

Jesús Labarta
BSC

Keynote @ IWOMP 2016
Nara, October 7th 2016
The multicore and memory revolution
- ISA leak …
- Plethora of architectures
  - Heterogeneity
  - Memory hierarchies

Complexity +
+ variability =
= Divergence …
- … between our mental models and actual system behavior

What programmers need? HOPE !!!
The programming revolution

An age changing revolution

– From the latency age …
  • I need something … I need it now !!!
  • Performance dominated by latency in a broad sense
    – At all levels: sequential and parallel
    – Memory and communication, control flow, synchronizations

– …to the throughput age
  • Ability to instantiate “lots” of work and avoid stalling for specific requests
    – I need this and this and that … and as long as it keeps coming I am ok
    – (Much broader interpretation than just GPU computing !!!)
  • Performance dominated by overall availability/balance of resources
Re-introduce “sény”
Decouple, insight
A quiet revolution

Program logic
independent of computing platform

General purpose
Task based
Concurrency + data

Intelligent runtime
Parallelization
Data management,
Dynamic resource management
Interoperability
Coordination with OS, …

Applications
PM: High-level, clean, abstract interface

Power to the runtime
ISA / API
History / Strategy

- SMPSs V1 ~2007
- CellSs ~2006
- GPUSs ~2009
- SMPSs V2 ~2009
- StarSs ~2008

- COMPSs ~2007
- COMPSs ServiceSs ~2010
- COMPSs ServiceSs PyCOMPSs ~2013

- PERMPAR ~1994
- GridSs ~2002
- NANOS ~1996
- DDT @ Parascope ~1992

- OpenMP
  - 3.0
  - 4.0
  - 5.0
  - 2008
  - 2013
  - 2017

- OmpSs ~2009
- OmpSs v2 ~2016

Forerunner of OpenMP
The StarSs family of programming models

Key concept
- **Sequential task based** program on single address/name space + directionality annotations
- Happens to execute parallel: Automatic run time computation of dependencies between tasks

Differentiation of StarSs
- **Dependences**: Tasks instantiated but not ready. Order IS defined
  - Lookahead
    - Avoid stalling the main control flow when a computation depending on previous tasks is reached
    - Possibility to “see” the future searching for further potential concurrency
  - Dependences built from data access specification
- Locality aware
  - Without defining new concepts
- Homogenizing heterogeneity
  - Device specific tasks but homogeneous program logic
The StarSs “Granularities”

- **StarSs**
  - @ SMP
  - @ GPU
  - @ Cluster

- **OmpSs**
- **COMPSs**
- **PyCOMPSs**

### Average task Granularity:
- **StarSs**: 100 microseconds – 10 milliseconds
- **COMPSs**: 1 second - 1 day

### Address space to compute dependences:
- **Memory** (StarSs)
- **Files, Objects (SCM)** (COMPSs)

### Language binding:
- **C, C++, FORTRAN** (StarSs)
- **Java, Python** (PyCOMPSs)

**Parallel** | **Ensemble, workflow**
OmpSs

Experimental platform
- Compiler, runtime, applications

Forerunner for OpenMP
- “extending” OpenMP
- “following” OpenMP

Minimalist set of concepts …
- … relaxing StarSs functional model
- … still looking for elegance and fundamentals
- … aggressively give “power to the runtime”
OmpSs in one slide

Minimalist set of concepts …

```
#pragma omp task [ in (array_spec, l_values...) ] [ out (...) ] [ inout (..., v[neigh[j]], j=0;n)] ]
    [ concurrent (...) ] [ commutative(...) ] [ priority(P) ] [ label(...) ]
    [ shared(...) ][ private(...) ][ firstprivate(...) ][ default(...) ][ untied ]
    [ final(expr)][ if (expression) ]
    [ reduction(identifier : list) ]
    [ resources(...) ]
{code block or function}
```

```
#pragma omp taskwait [ { in | out | inout } (…) ] [noflush]
```

```
#pragma omp taskloop [ grainsize(...) ] [ num_tasks(...) ][ nogroup ] [ in (…) ][ reduction(identifier : list) ]
{for_loop}
```

```
#pragma omp target device ({ smp | opencl | cuda })
    [ implements ( function_name ) ]
    [ copy_deps | no_copy_deps ] [ copy_in( array_spec ,…)] [ copy_out (…) ] [ copy_inout (…) ]
    [ndrange (dim, …)] [ shmem(…) ]
```

---

E. Ayguade, et al, “A Proposal to Extend the OpenMP Tasking Model for Heterogeneous Architectures” IWOMP 2009 & IJPP

A. Duran, et al, “Extending the OpenMP Tasking Model to Allow Dependent Tasks” IWOMP 2008, LNCS & IJPP
Follow OpenMP syntax

- For adopted OmpSs features
- Adapt semantics for OpenMP features. Ensure High compatibility

```plaintext
#pragma omp parallel // ignore

#pragma omp for [ shared(...)][private(...)][firstprivate(...)][schedule_clause] // ≈ taskloop
{for_loop}

#pragma omp task [depend (type: list)]
```
Dependences vs OpenMP

- **Regions**
  - Runtime versions that handle partial overlap.
  - Overhead but useful.

- **l_values**
  - Mechanisms to reintroduce size if used with regions

- **Commutative, concurrent**
  - relaxed inouts: possible reorders between those in a linear chain

- **Reductions**
  - Concurrent + privatization + associative & commutative operation

- **Multidependences**
  - Variable number of in/outs

\[
\text{in } (a[i:j])
\]

\[
\text{in } (*p)
\]

\[
\text{in } ([\text{size}]*p)
\]

\[
\text{in } (v[\text{neigh}[j]], j=0;n)
\]
void gs (float A[(NB+2)*BS][(NB+2)*BS])
{
    int it, i, j;

    for (it=0; it<NITERS; it++)
        for (i=0; i<N-2; i+=BS)
            for (j=0; j<N-2; j+=BS)
                gs_tile(&A[i][j]);
}

#pragma omp task
    in(A[0][1:BS], A[BS+1][1:BS], \
        A[1:BS][0], A[1:BS][BS+1]) \
    inout(A[1:BS][1:BS])
void gs_tile (float A[N][N])
{
    for (int i=1; i <= BS; i++)
        for (int j=1; j <= BS; j++)
            A[i][j] = 0.2*(A[i][j] + A[i-1][j] + \
                            A[i+1][j] + A[i][j-1] + \
                            A[i][j+1]);
}
```c
for (int j; j<N; j+=BS) {
    actual_size = (N - j > BS ? BS : N-j);
    #pragma omp task in(vec[j;actual_size]) inout(result)
    for (int count = 0; count < actual_size; count++, j++)
        result += f(&vec[j], actual_size);
}
# pragma omp task input (result)
printf(“TOTAL is %d\n”, result);
# pragma omp taskwait
```
for (int j; j<N; j+=BS){
    actual_size = (N- j > BS ? BS: N-j);
    
    #pragma omp task in(vec[j;actual_size]) concurrent(result)
    for (int count = 0; count < actual_size; count ++, j++) {
        #pragma omp atomic
        result += f(&vec[j], actual_size) ; } 
} 
#pragma omp task input (result)
printf ("TOTAL is %d\n", result);
#pragma omp taskwait
for (int j; j<N; j+=BS){
    actual_size = (N- j> BS ? BS: N-j);
    #pragma omp task in(vec[j;actual_size]) commutative(result)
    for (int count = 0; count < actual_size; count ++, j++)
        result += f(&vec[j], actual_size) ;
} 
#pragma omp task input (result)
printf ("TOTAL is %d\n", result);
#pragma omp taskwait

Tasks executed out of order but not concurrently

No mutual exclusion required
**Concurrent, Commutative**

**Flexibility in execution orders**
- Many other tasks can interleave
- Still maintain individual dependence relationships between tasks involved in the inout chain and the “outside world”
- May be interprocedural
Task reductions

- While-loops, recursions

```c
while (node) {  
    #pragma omp task \ 
    reduction(+: res)  
    res += node->value;  
    node = node->next;  
}  
#pragma omp task inout(res)  
printf("value: %d\n", res);  
#pragma omp taskwait
```


```c
#pragma omp parallel  
{  
    #pragma omp single  
    {  
        #pragma omp taskgroup \ 
        reduction(+: res) \ 
        firstprivate (node)  
        {  
            while (node) {  
                #pragma omp task \ 
                in_reduction(+: res)  
                res += node->value;  
                node = node->next;  
            }  
            printf("value: %d\n", res);  
        }  
    }  
}
```

Array reductions
- Typical pattern: reductions on large arrays with indirection

Implementation
- Privatization becomes inefficient when scaling cores and data size
- Atomics can introduce significant overhead
- PIBOR Proposal: Privatization with in-lined block-ordered reductions
  - Save footprint
  - Trade processor cycles for locality
- Generalization of implementations
  - Inspector executor based, …

```cpp
for (auto t : tasks) {
    #pragma task reduction (+:v[0:size]) \ private (j)
    for (auto i : taskIters) {
        j= f(i);
        v[j] += expression;
    }
} #pragma taskwait
```

Ciesko, J., et al. “Boosting Irregular Array Reductions through In-lined Block-ordering on Fast Processors”, HPEC15
Homogenizing Heterogeneity

ISA heterogeneity
Single address space program ... executes in several non coherent address spaces
- Copy clauses:
  - ensure sequentially consistent copy accessible in the address space where task is going to be executed
  - Requires precise specification of data accessed (e.g. array sections)
- Runtime offloads data and computation and manages consistency

Kernel based programming
- Separation of iteration space identification and loop body

```
#pragma omp target device ({ smp | opencl | cuda }) \ 
  [ copy_deps | no_copy_deps ] [ copy_in ( array_spec ,...)] [ copy_out (...)] [ copy_inout (...)] \ 
  [ implements ( function_name )] \ 
  [shmem(...) ]\ 
  [ndrange (dim, g_array, l_array)]
```

```
#pragma omp taskwait [ on (...) ][noflush]
```
Automatic memory management and kernel synchronization

```c
#pragma target device (smp) copy_deps
#pragma omp task input ([size] c) output ([size] b)
void scale_task (double *b, double *c, double scalar, int size)
{
    for (int j=0; j < size; j++) b[j] = scalar*c[j];
}

#pragma target device (cuda) copy_deps nrange(1, size, 128)
#pragma omp task input ([size] c) output ([size] b)
__global_ void scale_task_cuda (double *b, double *c, double scalar, int size);
```

```c
double A[1024], B[1024], C[1024]
double D[1024], E[1024];
main(){
    ...
    scale_task_cuda(A, B, 10.0, 1024);  //T1
    scale_task_cuda(B, A, 0.01, 1024);  //T2
    scale_task (C, A, 2.0, 1024);      //T3
    scale_task_cuda (D, E, 5.0, 1024);  //T4
    scale_task_cuda (B, C, 3.0, 1024);  //T5
#pragma omp taskwait
    // can access any of A,B,C,D,E
}
```
Homogenizing Heterogeneity

```c
#pragma omp target device(opencl) ndrange(1,size,128) copydeps implements (calculate_forces)
#pragma omp task out([size] out) in([npart] part)
__kernel void calculate_force_opencl(int size, float time, int npart, __global Part* part,
        __global Part* out, int gid);

#pragma omp target device(cuda) ndrange(1,size,128) copydeps implements (calculate_forces)
#pragma omp task out([size] out) in([npart] part)
__global__ void calculate_force_cuda(int size, float time, int npar, Part* part, Particle *out, int gid);

#pragma omp target device(smp) copydeps
#pragma omp task out([size] out) in([npart] part)
void calculate_forces(int size, float time, int npart, Part* part, Particle *out, int gid);

void Particle_array_calculate_forces(Particle* input, Particle *output, int npart, float time) {
    for (int i = 0; i < npart; i += BS )
        calculate_forces(BS, time, npart, input, &output[i], i);
}
```
MACC (Mercurium ACcelerator Compiler)

“OpenMP 4.0 accelerator directives” compiler
- Generates OmpSs code + CUDA kernels (for Intel & Power8 + GPUs)
- Propose clauses that improve kernel performance

Extended semantics
- Change in mentality … minor details make a difference
- Dynamic parallelism

G. Ozen et al, “On the roles of the programmer, the compiler and the runtime system when facing accelerators in OpenMP 4.0” IWOMP 2014

G. Ozen et al, “Multiple Target Work-sharing support for the OpenMP Accelerator Model” submitted
Interoperability: MPI and OmpSs

Taskifying MPI calls

Potential issues
- Deadlocks if blocking resources
  - virtualize MPI engine
  - Nanos6 on pthreads, argobots,…
- Matching if executed out of order

Throughput oriented Opportunity
- Overlap between phases
  - Grid and frequency domain
- Provide laxity for communications
  - Tolerate poorer communication
- Shift load balance issue
  - Eliminate serialization
  - Increase granularity
- Huge flexibility for changing behavior with minimal syntactic changes


IFS weather code kernel. ECMRWF

physics

ffts
Hybrid Amdahl’s law

"A fairly “bad message” for programmers"

"Significant non parallelized part"

– MPI calls + pack/unpack

"MPI + OmpSs: Hope for lazy programmers"
MPI offload

```c
MPI_Comm comm_slaves, comm_workers;

void processShot(int i, shot_handler_t* shot)
{
    read_shot(i, shot); // Read shot from GPFS
    int page = shot->size/4;
    for(int p0, p4, ++j++)
    {
        float *data = shot->data[page];
        #pragma omp task inout(data, page) onto(comm_workers, j)
        processShotSlice(data, page);
    }
    #pragma omp taskwait
}

int main(int argc, const char* argv[])
{
    int n_slaves = 256, nshots = 32;
    shot_handler_t shots[nshots];
    // Master nodes allocate slaves 256 slaves
    deep_boost_alloc(MPI_COMM_SELF, 16, 16, &comm_slaves);
    for(int i=0; i<n_slaves; i++)
    {
        // Each slave allocates 4 workers
        #pragma omp task onto(comm_slaves, i)
        deep_boost_alloc(MPI_COMM_SELF, 4, 1, &comm_workers);
    }
    for(int i=0; i<nshots; i++)
    {
        #pragma omp task inout(shots[i]) onto(comm_slaves)
        {
            processShot(i, shots[i]);
            merge_and_save_shot(shots[i]); // GPFS I/O intensive
        }
        #pragma omp taskwait
    }
}
```

CASE/REPSOL FWI

Reverse offload

Speedup of FWI using 4 nodes per worker group

![Graph showing speedup of FWI using 4 nodes per worker group](image)
COMPILER AND RUNTIME
Locality aware scheduling
- Affinity to core/node/device can be computed based on pragmas and knowledge of where was data
- Following dependences reduces data movement
- Interaction between locality and load balance (work-stealing)

Some “reasonable” criteria
- Task instantiation order is typically a fair criteria
- Honor previous scheduling decisions when using nesting
  • Ensure a minimum amount of resources
  • Prioritize continuation of a father task in a taskwait when synchronization fulfilled

Criticality-awareness in heterogeneous architectures

- **Heterogeneous multicores**
  - ARM big.LITTLE 4 A-15@2GHz; 4A-7@1.4GHz
  - Tasksim simulator: 16-256 cores; 2-4x

- **Runtime approximation of critical path**
  - Implementable, small overhead that pay off
  - Approximation is enough

- **Higher benefits the more cores, the more big cores, the higher performance ratio**

Implicit specification and automatic management (transfers, caching, coherence)

Automatic association management
- Workarrays & Reshaping

```c
#pragma css task input(T{0:BS}{0:BS}, BS, N) inout(B{0:BS}{0:BS})
void strsm_tile(integer BS, integer N, float T[N][N], float B[N][N]) {
    unsigned char LO='L', TR='T', NU='N', RI='R';
    float DONE=1.0;
    integer LDT = sizeof(*T)/sizeof(float);
    integer LDB = sizeof(*B)/sizeof(float);
    strsm_(&RI, &LO, &TR, &NU, &BS, &BS, &DONE, T, &LDT, B, &LDB);
}
```

Using MKL kernels/tiles
Highly NUMA machine

- Asynchrony with serialized initialization

- Effect of parallel initialization and first touch

- Copy to workarray. Change of association

- NUMA aware workarray allocation
FPGAs

Just another heterogeneous device

Experiments @ Zynq

```c
#pragma omp target device(smp, fpga) copy_deps
#pragma omp task in([Dim*Dim]A, [Dim*Dim]B) inout([Dim*Dim]C)
void Mm_Kernel(T a[Dim][Dim], T b[Dim][Dim], T out[Dim][Dim]) {
    #pragma HLS inline
    #pragma HLS array_partition variable=a block factor=Dim/2 dim=2
    #pragma HLS array_partition variable=b block factor=Dim/2 dim=1
    for (int ia = 0; ia < Dim; ++ia) {
        for (int ib = 0; ib < Dim; ++ib) {
            #pragma HLS pipeline
            T sum = out[ia][ib];
            for (int id = 0; id < Dim; ++id) {
                sum += a[ia][id] * b[id][ib];
                out[ia][ib] = sum;
            }
        }
    }
}
```
Device management mechanisms

- Improvements in runtime mechanisms
  - Use of **multiple streams**
  - High asynchrony and overlap (transfers and kernels)
  - Overlap kernels
  - Take overheads out of the critical path

- Improvement in schedulers
  - Late binding of locality aware decisions
  - Propagate priorities

**Nbody**

**Cholesky**
Dynamic Load Balancing

Dynamically shift cores between processes in node
- Enabled by application malleability (task based)
- Runtime in control (sees what happens and the future)
- Would greatly benefit from OS support (handoff scheduling, fast context switching)

“LeWI: A Runtime Balancing Algorithm for Nested Parallelism”. M.Garcia et al. ICPP09
THE REAL REVOLUTION
The real revolution

- Task based
  - OpenMP OK

- “Proper” model does not guarantee “proper” programs
  - Flexibility → can be used “wrong”
  - Possible to write an MPI program in OpenMP syntax

- Revolution: is in the mindset of programmers
  - “Forget” about hardware, resources
    - rely on the runtime – system
  - Focus on program logic
  - Methodology
    - Top down programming methodology
    - Throughput oriented:
      - try not to stall!
      - First order, then overhead
    - Think global:
      - of potentials rather than how-to’s
      - may be unprecise
    - Specify local:
      - needs and outcomes of the functionality being written
      - precise
### Programming practices

#### What to avoid
- Threads
  - Omp_num_threads
  - Thread_private
  - Parallel, barrier
- separate parallel and serial implementation
- Ifdefs
- Infer too much from scaling plots
- Worry too early about actual performance

#### What to try
- Top down & nesting
- Lookahead
  - synchronizing tasks, handle control flow dependences
- Malleability
- Taskify communications
  - Overlap computation, other communications, shift critical path

```c
foo() {
  if (small) for(;;) {...};
  else {
    #pragma omp parallel for
    for(;;) {...};
  }
}
```
Examples

Applications
- PARSECS
- Nt-chem
- Alya
- Lulesh
- IFSkernel
- Quantum Expreso
PARSEC benchmark ported to OmpSs

Initial port from pthreads to OmpSs and optimization

D. Chasapis et al., “Exploring the Impact of Task Parallelism Beyond the PARSEC benchmark suite” TACO’2016
Electronic structure calculation
RIKEN FIBER miniapp

imp2_rmp2energy_incore_v_mpiomp.F90

1:   RIMP2_RMP2Energy_InCore_V_MPIOMP ()

405:   DO LNumber_Base

498:       DGEMM

518:           if (something)

588:               Do loops

636:               END DO

ENDO
Original: Hybrid MPI + OpenMP

- Not fully populated node → system activates TurboBoost increasing frequency from 2.6 to 3.1GHz
- Load imbalance
- Global Serialization
- Noise
- Some gain @ low threading count
- High overhead, fine granularity @ large threading count
Load imbalance

- Global
- Migrating load imbalance, Serialization

Asynchrony: non blocking MPI calls
mn3 : OmpSs taskification

Taskify

- Serial DGEMMs.
  - Coarser granularity than original OpenMP
- Reduction loops
  - Not parallelized in original?
  - Serial task
- Communications
  - Overlap, but fixed schedule in original source

Outcome

- Possible with limited understanding of global application
- Happen to be fairly independent

imp2_rmp2energy_incore_v_mpiomp.F90

1: RIMP2_RMP2Energy_InCore_V_MPIOMP ()
...
405: DO LNumber_Base
498: DGEMM
518: if (something) {
  wait ;
  Isend, Irecv ;
} // for current iter.
588: allreduce
Do loops
Evaluating MP2 correlation
636: END DO
ENDO
Performance gain

- Sufficient task granularity
- Communication computation overlap
- Still measuring numbers with DLB
Analyses

- Trace MPI + OmpSs, 4 x 4
  - Stalls:
    - Interaction throttling ↔ scheduling ↔ loose dependence chains
    - Prioritize communication tasks

- Concurrent MPI tasks →
  - ~ commutative send/rec tasks
    • only one thread executes MPI, avoid contention …
    • … but still contention with allreduces!

- Task generation overhead →
  - Reduce number of tasks
    • Separate send and receive tasks → MPI_send_rec
    • Increase DGEMM size, reduce #tasks (WIP)
FE + Particles

Important imbalance
- ~ unpredictable, not fixable at domain decomposition level

By hybrid programming
- Reduce processes $\rightarrow$ less imbalance
  - Sequential performance?
  - “Hybrid Amdahl’s law”?  
- Still imbalance
Matrix Assembly

Reduction on large matrix with indirection

```c
#pragma omp parallel for
compute()
#pragma omp atomic update()
}
```
Sequential performance

Parallelization of indirect reduction on large object

- Impact on IPC
- Still load imbalance

- atomics
- coloring
- Commutative multideps
- + priority
Dynamic load balance

**DLB**
- Across MPI processes
- Within a node
- 14% gain (38.5% over coloring)

**Side effects**
- Frequently imbalance concentrated in contiguous processes
- Suggestion:
  - Interleave processes
- Result:
  - Better Pure MPI performance !!!
Processes & Threads

Throughput computing
- Malleability (tasks) + DLB → flat lines

DLB helps in all cases
- Even more in the bad ones 😊

Side effect
- Hybrid Nx1 can be better than pure MPI !!!

Assembly phase

Subscale phase
Granularity

- Fairly wide range of good granularities

→ Throughput computing
Other features

- Memory hierarchy management
- Nesting/recursion
- Criticality and locality aware scheduling
- Multiple implementations for a given task
- CUDA, OpenCL, accelerator directives, FPGA, Cluster
- Resilience
- Real time
- ...

Commitment: forerunner for OpenMP

- Continuous development and use since 2004
- Pushing ideas into the OpenMP standard
- Developer Positioning: efforts in OmpSs will not be lost.
What is important? Methodology

- The revolution requires a programming effort
  - Must make the transition it as simple as possible
  - Must make it as long lived as possible

- Top down !!
  - Every level contributes
  - Nesting
  - Think global, specify local

- Throughput oriented, asynchrony, do not stall
  - Granularity: stay within large plateau of “good” performance
  - Try to avoid predefined schedule of synchronizing operations
  - Try to avoid “machine” specificities

- First order and flexible decomposition, then overhead

- Malleability / Responsiveness

- Incremental:
  - Only where needed (e.g only taskify to enable DLB, overlap,…)

- Show your code, look at others code, don’t get just dazzled by performance

- Do not fly blind
  - Very aggregated statistics may not be enough to gain the insight on actual behavior
The real parallel programming revolution …

… is in the mindset of programmers

… is to rely on the runtime and system

https://pm.bsc.es/ompss-downloads
Contributors

- Eduard Ayguade
- Rosa M. Badia
- Xavier Martorell
- Vicenç Bertran
- Alex Duran (Intel)
- Roger Ferrer (ARM)
- Xavier Teruel
- Javier Bueno (Metempsy)
- Judit Planas (Lausanne)
- Sergi Mateo
- Carlos Alvarez
- Daniel Jimenez-Gonzalez
- Guillermo Miranda (UPC)
- Diego Caballero (Intel)
- Jorge Bellon
- Antonio Filgueras
- Florentino Sainz (BBVA)
- Diego Nieto (...)
- Victor Lopez
- Marta Garcia
- Josep M. Perez
- Guray Ozen
- Antonio J. Peña
- Julian Morillo
- Sara Royuela
- Marc Josep
- Miquel Vidal
- Kevin Sala
- Marc Mari
- Aimar Rodriguez
- Daniel Peyrolon
- Ferran Pallares
- Albert Navarro
- Toni Navarro
- Omer Subasi
- Jan Ciesko
- Marc Casas
- Miquel Moreto
- …