The Parallel Programming Osmotic Membrane

Prof. Jesús Labarta
BSC & UPC

44ème Forum ORAP
Paris, Nov. 29th 2019
Code lifetime cycle

- Need
- Interest
- Idea
- Model

Code

Code

“Platform/language specificities”
“Optimizations”
“Hardwired order/schedules”

Code

Code

Code
There is HOPE !!!

Need
Interest
Idea
model

Code

Code

Code

Code

Performance portability
Maintainable, adaptable
Focus on logic

Code
The PM osmotic membrane

Applications

PM: High-level, clean, abstract interface

Power to the runtime

ISA / API

General purpose
Task & data based
Forget about resources
Decouple: Minimal & sufficient permeability?

Intelligence & Resource management
“Reuse & expand” old architectural ideas under new constraints
Integrate concurrency and data

- Single mechanism
  - Concurrency:
    - Dependences built from data accesses
    - Lookahead: About instantiating work
  - Locality & data management
    - From data accesses

```c
void Cholesky(int NT, float *A[NT][NT]) {
    for (int k=0; k<NT; k++) {
        #pragma omp task inout ([TS][TS](A[k][k]))
        spotr (A[k][k], TS);
        for (int i=k+1; i<NT; i++) {
            #pragma omp task in([TS][TS](A[k][k])) inout ([TS][TS](A[k][i]))
            stram (A[k][k], A[k][i], TS);
        }
        for (int i=k+1; i<NT; i++) {
            for (j=k+1; j<i; j++) {
                #pragma omp task in([TS][TS](A[k][i]), [TS][TS](A[j][i])) inout ([TS][TS](A[j][i]))
                sgemm (A[k][i], A[k][j], A[j][i], TS);
            }
            #pragma omp task inout ([TS][TS](A[k][i]) inout([TS][TS](A[i][i]), TS));
        }
    }
}
```
OmpSs

• A forerunner for OpenMP
The real revolution

• Hybrid Task based ~ OK, but ...

• “Proper” model does not guarantee “proper” programs
  • Flexibility → can be used “wrong”
  • Legacy features have a strong “shadow”

• Revolution is in the mindset of programmers
  • “Forget” about hardware, resources
  • Focus on program logic
  • Methodology
    • Top down programming methodology
      • Every level contribute
    • Throughput oriented
      • try not to stall!
  • Think global:
    • may be unprecise
  • Specify local:
    • Precise

To exascale ... and before.

And it IS disruptive !!!
Performance Optimisation and Productivity
A Centre of Excellence in Computing Applications

Contact:
https://www.pop-coe.eu
mailto:pop@bsc.es

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 676553.
Some recommendations
“Recommendations”

• Don’t mask your symptoms … try to understand
  • Taskify with dependences … do not stall !!!!
    • Computation
    • communications

• Do nest … everybody's contribution counts
  • Top down
  • Precise region specification

• Do hint … do not force !!!

• Think malleable …. be water my friend

• Homogenize heterogeneity

• Beware of reductions on large arrays

• ...

... all of this can be achieved in an incremental way
Don’t mask your symptoms ...

https://pop-coe.eu
... try to understand
... try to understand

IFS-SP
420x4

Serialized unpacking

Serialized communication pattern

Very fine grain parallelization of individual unpacks
Don’t mask your symptoms ...

```
Need
Interest
Idea
model
```

```
Code
```

```
Code
```

```
Code
```

```
Code
```

```
Code
```

```
“Platform/language specificities”
“Optimizations”
“Hardwired order/schedules”
```

```
Where is the undo button ??
```
Taskify computation

Four loops/routines
Sequential program order

Fork join parallelism
not parallelizing one loop

“A chance for lazy programmers”

Task based, top down parallelism
not parallelizing one loop and still fine

GROMACS
Taskify communication

- MPI + OpenMP
- Non blocking MPI + OpenMP
- MPI + OmpSs
  - Top down
  - Overlap communications
  - Serial FFTs
  - Replicate communicators

FFTlib - Quantum Espresso mini app
1:  RIMP2_RMP2Energy_InCore_V_MPIOMP ()

... DO LNumber_Base

498:  DGEMM

... if (something)
{ wait ; // for current iter.
  Isend, Irecv ; // for next iter.
}

allreduce

518:  Do loops

Evaluating MP2 correlation

588:  END DO

636:  ENDO

NTCHEM

Don’t mask your symptoms

Top down

Leave order (overlap) to OpenMP
Do nest

- Nest tasks, not parallels
- Dynamic over decomposition

for (int R = 1; R < rows-1; R += bs) {
    #pragma oss task weakin(matrix[R-1][1;cols-2] \ 
    matrix[R+bs][1;cols-2]) weakinout(matrix[R;bs][1;cols-2])
    for (int C = 1; C < cols-1; C += bs) {
        #pragma oss task in(matrix[R-1][C;bs], matrix[R;bs][C-1], \ 
        matrix[R;bs][C+bs]) inout(matrix[R;bs][C;bs])
        for (int r=R; (r<rows-1) && (r<R+bs); r++)
            for (int c=C; (c<cols-1) && (c<C+bs); r++)
                matrix[r][c] = 0.25 * (matrix[r][c] + matrix[r][c] + matrix[r][c] + matrix[r][c]);
    }
}

Do nest

```c
for ( d = 0; d < dt; d++){
    for ( c = 0; c < ct; c++){
        #pragma oss task ...
        dtrsm( ... );
    }
    decide(final_1);
    for ( c = 0; c < ct; c++) {
        #pragma oss task weakinout (TILEB[d:rt-1][c]) ... final(final_1)
        for ( r = d; r < rt; r++)
            decide(inal_2);
        #pragma oss task inout(TILEB[r][c]) ... final(final_2)
        dgemm( ... );
    }
}
```
Do Hint

• Priorities
• Anti-dependence distances

Homogenize Heterogeneity

• Performance heterogeneity

• ISA heterogeneity

• Several non coherent address spaces
Think malleable

• Dynamic Load Balance & Resource management
  • Intra/inter process/application

• Library (DLB)
  • Runtime interception (MPIP, OMPT, ...)
  • API to hint resource demands
  • Core reallocation policy

• Opportunity to fight Amdalh’s law
  • Productive / Easy !!!
    • Nx1
    • Hybridize imbalanced regions

“LeWI: A Runtime Balancing Algorithm for Nested Parallelism”, M.Garcia et al. ICPP09
“Hints to improve automatic load balancing with LeWI for hybrid applications” JPDC2014
Think malleable

• Dynamic Load Balance & Resource management
  • Intra/inter process/application

• Library (DLB)
  • Runtime interception (MPIP, OMPT, ...)
  • API to hint resource demands
  • Core reallocation policy

• Opportunity to fight Amdalh’s law
  • Productive / Easy !!!
    • Nx1
    • Hybridize imbalanced regions

“LeWI: A Runtime Balancing Algorithm for Nested Parallelism”. M.Garcia et al. ICPP09
“Hints to improve automatic load balancing with LeWI for hybrid applications” JPDC2014
Beware of reductions on large arrays

- Reductions with indirection on large arrays with indirections
  - Atomic
  - Array privatization + final reduction
  - Coloring
  - Serialize
    - Commutative clause
  - Specify incompatibilities !!!
    - Commutative + multidependences
At all levels
At all levels

StarSs

OmpSs

@ SMP  @ GPU  @ FPGA  @ Cluster

Average task Granularity:

 microseconds  milliseconds

Address space to compute dependences:

Memory

Language binding:

C, C++, FORTRAN

Core  Parallel  Ensemble, workflow
At all levels

StarSs

(long) vectors

@ vector ISA

OmpSs

@ SMP
@ GPU
@ FPGA
@ Cluster

COMPSs

PyCOMPSs

@ Multicore
@ Cluster
@ Cloud

Average task Granularity:
nanoseconds microseconds milliseconds seconds hours days

Address space to compute dependences:
Registers Memory Objects Files

Language binding:
C, pragmas/intrinsics C, C++, FORTRAN Java, Python

Core Parallel Ensemble, workflow
PyCOMPSs

Main program

```
num_frags = ...
todo_list = init_list();
for i in range(num_frags):
    result = cc_sur(todo_list[i])
    gather(result, global_result)
compss_stop()
```

Tasks definition

```
@task(returns = list, result = IN, global_result = INOUT))
def gather(result, global_result):
@task(returns = list)
def cc_sur():
```
RISC-V VECTOR COMPILER

- Programming → RISC-V Vector extension ISA
  - Intrinsics (https://repo.hca.bsc.es/gitlab/rferrer/epi-builtins-ref/blob/master/README.md#documentation)
  - Pragma omp simd
  - Automatic parallelization

- Compiler
  - LLVM based

- Compiler Explorer (http://repo.hca.bsc.es/epic)
  - Interactive impact of source code modifications → code generated
  - A few example RISC-V vector codes using intrinsics
RISC-V VECTOR EMULATOR

- Emulation of RISC-V Vector ISA
  - Parametrized MAXVL

- Simple Timing models
  - Memory architecture
  - Instruction timing

- Very detailed analytics in Paraver → co-design
  - Vector length
  - Register use
  - Memory addresses, cache ratios
  - Instruction timing
  - ...

```
for (int kk = 0; kk < n; kk += bk) {
    vb0 = __builtin_epi_vload_f64(&b[kk][jj]);
    vb1 = __builtin_epi_vload_f64(&b[kk+1][jj]);
    vb2 = __builtin_epi_vload_f64(&b[kk+2][jj]);
    vb3 = __builtin_epi_vload_f64(&b[kk+3][jj]);
    __epi_f64 vta0, vta1, vta2, vta3;
    __epi_f64 vtp0, vtp1, vtp2, vtp3;
    vta0 = __builtin_epi_vbroadcast_f64(a[ii][kk]);
    vtp0 = __builtin_epi_vfmul_f64(vta0, vb0);
    vc0 = __builtin_epi_vfadd_f64(vc0, vtp0);
```

Timing model

Paraver

LLVM

Emulation library

trace2prv

Emulation environment

.MUSA
RISC-V VECTOR VISUALIZATION

- FE mockup

Program Counter

Memory address

Memory instruction cost
Final thoughts ...
Age before beauty

- Behavior (insight/models) before syntax
- Detail performance analytics before aggregated profiles
- Work instantiation and order before overhead
- Malleability before fitted rigid structure
- Possibilities before how tos
- Elegance before one day shine

- All about programmer mindset !!!

El abuelo cebolleta ataca de nuevo
Performance Optimisation and Productivity
A Centre of Excellence in Computing Applications

Contact:
https://www.pop-coe.eu
mailto:pop@bsc.es

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 676553.
Thanks!