www.bsc.es www.upc.edu

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Developing Applications with OmpSs@FPGA

Xavier Martorell
Universitat Politècnica de Catalunya, and
Barcelona Supercomputing Center

3rd Int. Workshop on FPGA for HPC
Tokyo, Japan - March 12th, 2018

Motivation

Heterogeneous architectures have come to stay

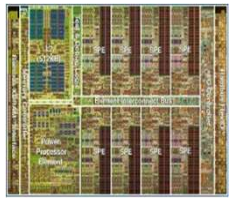
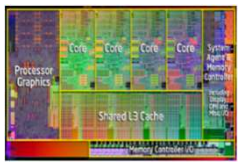
Applications

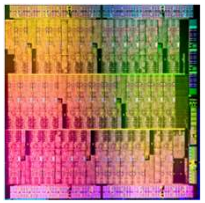
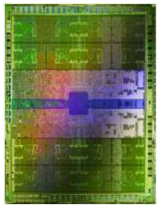
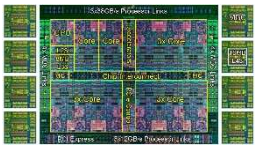
ISA / API

Direct Accesses to Architectural Details

- SIMD
- Local memories / data transfers
- Code execution

OmpSs: can we provide a consistent view, including these heterogeneous architectures?

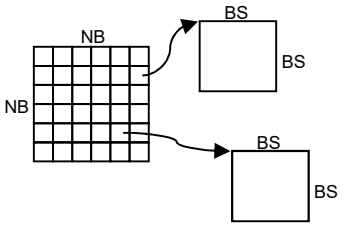
Matrix multiplication

```
#define BS 128

void matrix_multiply(float a[BS][BS], float b[BS][BS],float c[BS][BS])
{
// matrix multiplication of two A, B matrices, to accumulate the result on C
  for (int ia = 0; ia < BS; ++ia)
    for (int ib = 0; ib < BS; ++ib) {
      float sum = 0;
      for (int ic = 0; ic < BS; ++ic)
        sum += a[ia][ic] * b[ic][ib];
      c[ia][ib] += sum;
    }
}
```

Kernel

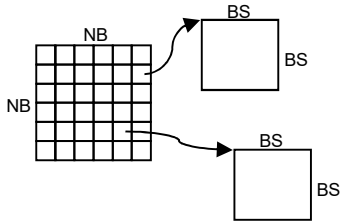
```
…
for (i=0; i<NB; i++)
  for (j=0; j<NB; j++)
    for (k=0; k<NB; k++)
      matrix_multiply(A[i][k], B[k][j], C[i][j]);
…
```

Main program

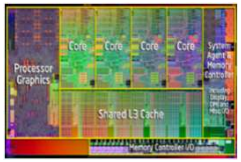


Heterogeneous programming

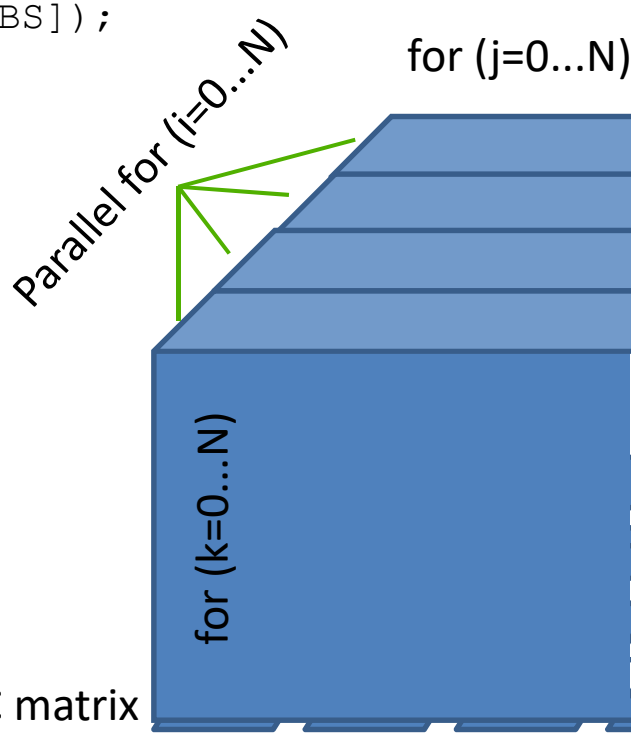
Homogenizing the support




```
void matrix_multiply(float a[BS][BS], float b[BS][BS],
                                      float c[BS][BS]);
…
#pragma omp parallel for private(j,k)
for (i=0; i<NB; i++)
  for (j=0; j<NB; j++)
    for (k=0; k<NB; k++)
      matrix_multiply(A[i][k], B[k][j], C[i][j]);
…
```

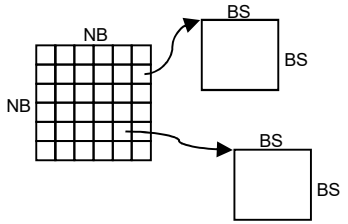


SMP

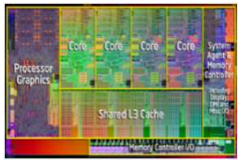


Heterogeneous programming

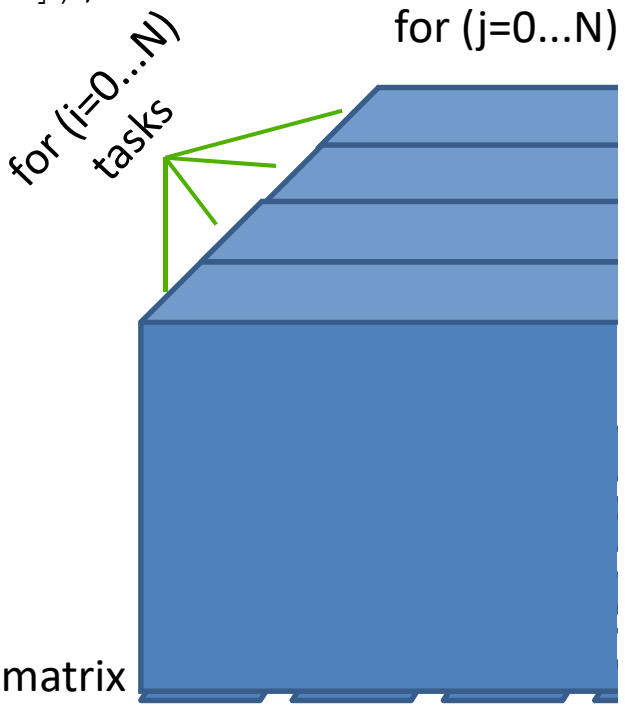
Homogenizing the support




```
void matrix_multiply(float a[BS][BS], float b[BS][BS],
                                      float c[BS][BS]);
…
for (i=0; i<NB; i++)
  #pragma omp task private(j,k)
  for (j=0; j<NB; j++)
    for (k=0; k<NB; k++)
      matrix_multiply(A[i][k], B[k][j], C[i][j]);
…
```



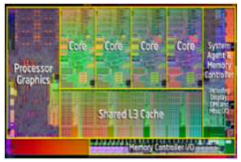
SMP



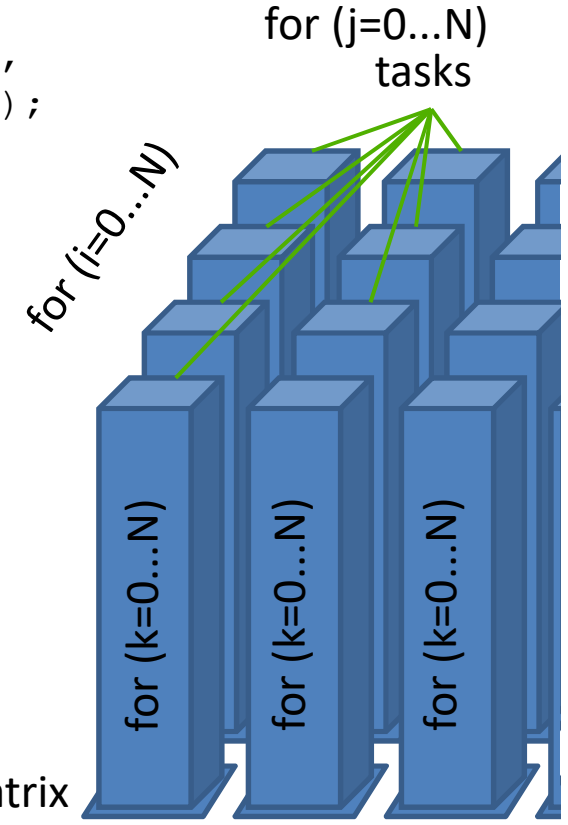
Heterogeneous programming

Homogenizing the support

```
void matrix_multiply(float a[BS][BS], float b[BS][BS],
                                      float c[BS][BS]);

…
for (i=0; i<NB; i++)
  for (j=0; j<NB; j++)
    #pragma omp task private(k)
    for (k=0; k<NB; k++)
       matrix_multiply(A[i][k], B[k][j], C[i][j]);
…
```



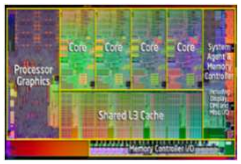
SMP



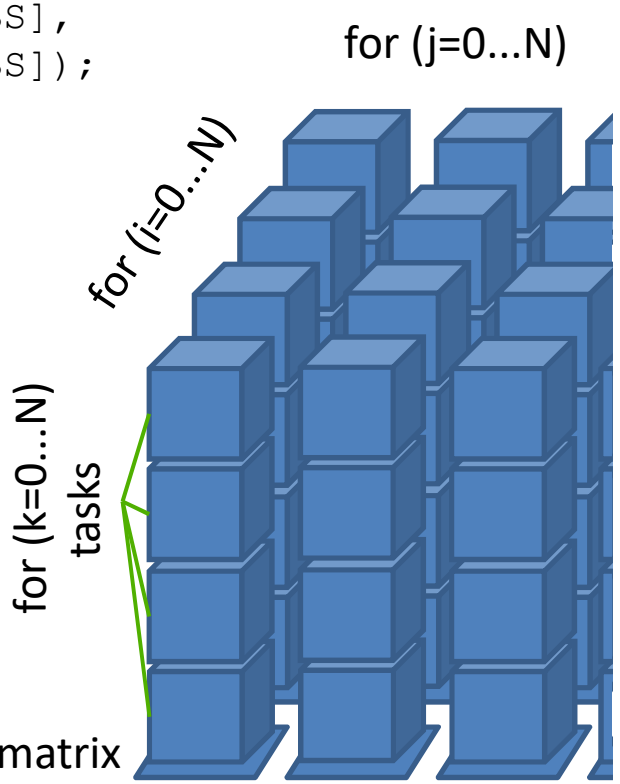
Heterogeneous programming

Homogenizing the support

```
void matrix_multiply(float a[BS][BS], float b[BS][BS],
                                      float c[BS][BS]);
…
for (i=0; i<NB; i++)
  for (j=0; j<NB; j++)
    for (k=0; k<NB; k++)
      #pragma omp task \
                  depend(in:A[i][k],B[k][j]) \
                  depend(inout:C[i][j])
      matrix_multiply(A[i][k], B[k][j], C[i][j]);
…
```



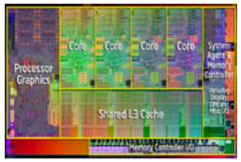
SMP



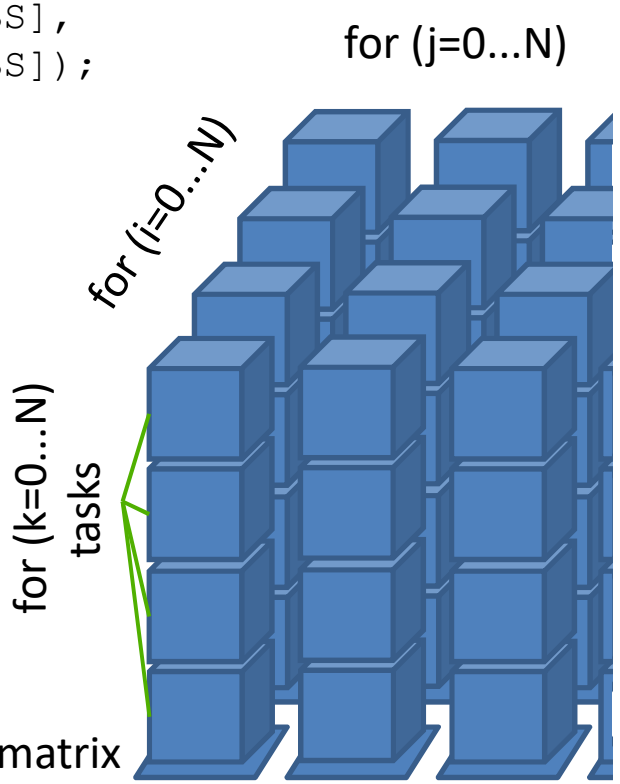
Heterogeneous programming

Homogenizing the support

```
#pragma omp task depend(in:a,b) depend (inout:c)
void matrix_multiply(float a[BS][BS], float b[BS][BS],
                                      float c[BS][BS]);
…
for (i=0; i<NB; i++)
  for (j=0; j<NB; j++)
    for (k=0; k<NB; k++)
      matrix_multiply(A[i][k], B[k][j], C[i][j]);
…
```



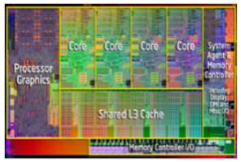
SMP



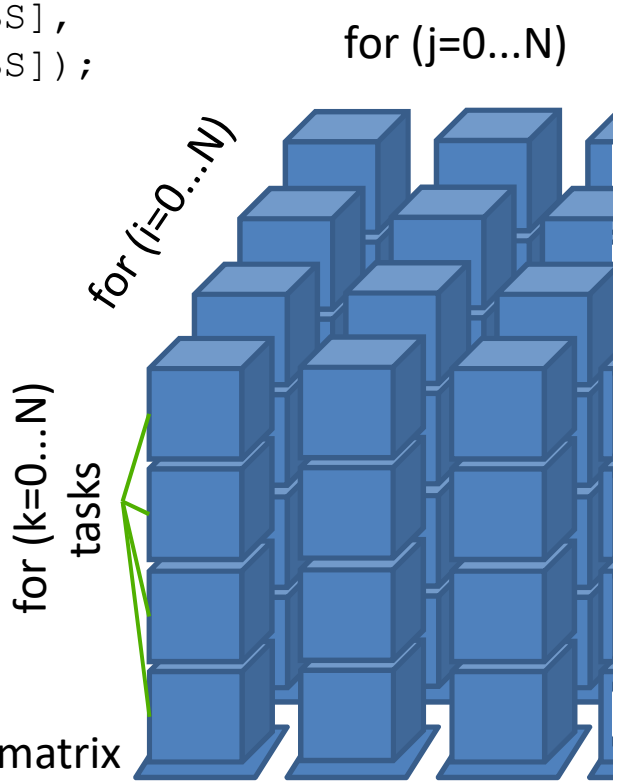
Heterogeneous programming

Homogenizing the support

```
#pragma omp target device(smp)
#pragma omp task depend(in:a,b) depend (inout:c)
void matrix_multiply(float a[BS][BS], float b[BS][BS],
                                      float c[BS][BS]);

…
for (i=0; i<NB; i++)
  for (j=0; j<NB; j++)
    for (k=0; k<NB; k++)
      matrix_multiply(A[i][k], B[k][j], C[i][j]);
…
```



SMP



All is about interfaces

int ol_main_par (**int low_i, int_high_i**, float *A, float *B, float *C)
- With private j, k

int ol_main_task (**int i**, float * A, float * B, float * C)
- With private j, k

int ol_main_task (**int i, int j**, float * A, float * B, float * C)
- With private k

int ol_main_task (**int i, int j, int k**, float * A, float * B, float * C,
dependence_descriptor_t * deps)
- No private variables

All is about interfaces

In previous slide, variables move from private to function args
- As programmers, we do not want to keep doing so on our applications!!

The compiler is the proper place to do these transformations
- Algorithm has been used in OpenMP over the past 20 years
- There is no magic with it

Avoid having to take into account these details from high level programming
- Let's remind that **in heterogeneous systems...**
  - » the call to these interfaces is done on the host code
  - » and the target is the accelerator code

Changes are so much ERROR prone!!!

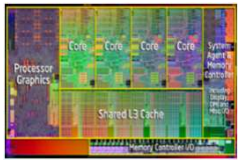
Heterogeneous programming

The “implements” approach

- Kernel provided in CUDA
- Kernel compiled “offline”
- Data transfers automatically generated by OmpSs

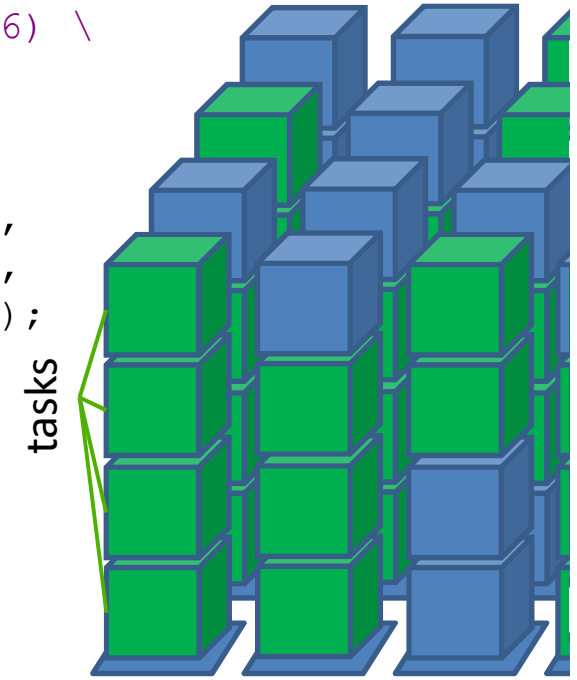
Single point of change

```
#pragma omp target device(cuda) ndrange(2,NB,NB,16,16) \
                implements(matrix_multiply)
#pragma omp task depend(in:a,b) depend (inout:c)
__global__ void matrix_multiply_cuda(float a[BS][BS],
                                     float b[BS][BS],
                                     float c[BS][BS]);
```



SMP

GPGPU



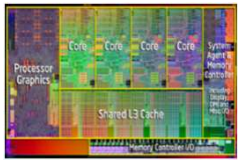
Heterogeneous programming

The “implements” approach

- Kernel provided in OpenCL
- Kernel compiled “online”
- Data transfers automatically generated by OmpSs

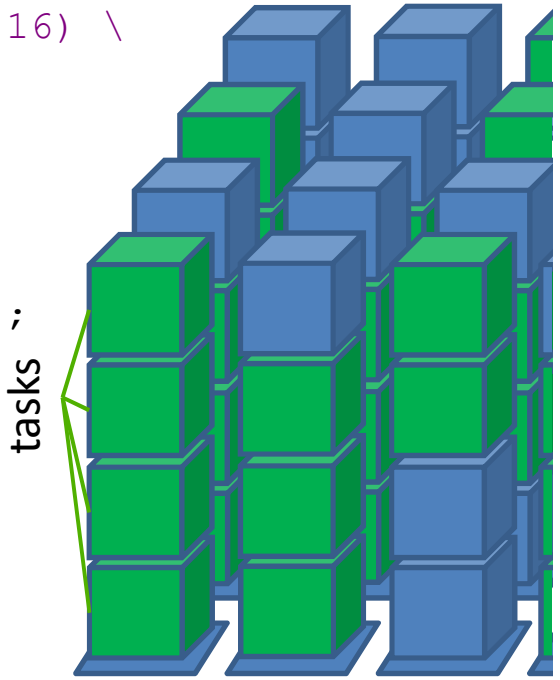
Single point of change

```
#pragma omp target device(opencl) ndrange(2,NB,NB,16,16) \
                  implements(matrix_multiply)
#pragma omp task depend(in:a,b) depend (inout:c)
__kernel void matrix_multiply_opencl(float a[BS][BS],
                                     float b[BS][BS],
                                     float c[BS][BS]);
```



SMP

GPGPU



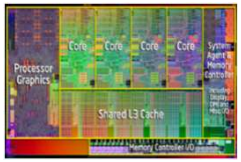
Heterogeneous programming

The “implements” approach

- Native C/C++ kernel!!! => Vivado HLS
- Kernel compiled “offline”... takes some time
- Data transfers automatically generated by OmpSs

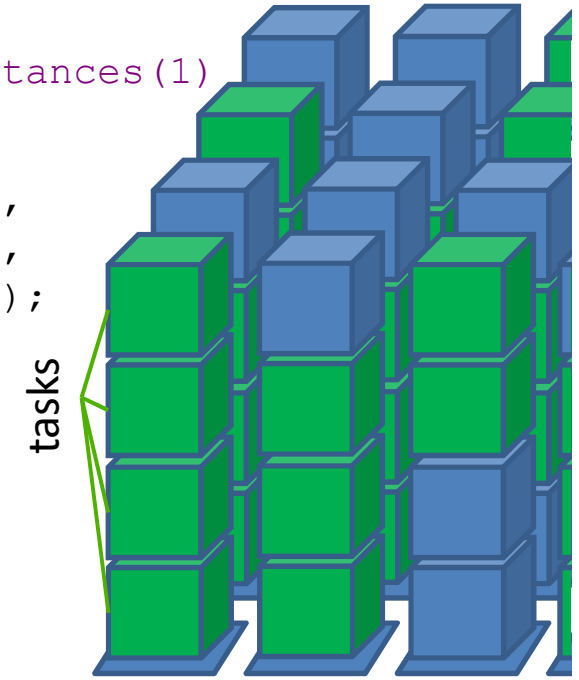
Single point of change

```
#pragma omp target device(fpga) \
                implements(matrix_multiply) //num_instances(1)
#pragma omp task in(a,b) inout(c)
          void   matrix_multiply_fpga(float a[BS][BS],
                                      float b[BS][BS],
                                      float c[BS][BS]);
```



SMP

FPGA



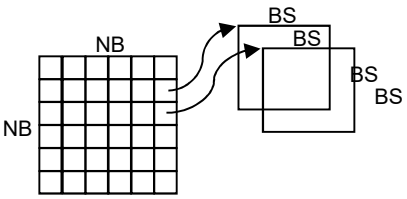
Heterogeneous programming

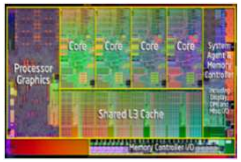
The “implements” approach

- Native C/C++ kernel!!! => Vivado HLS
- Kernel compiled “offline”... takes some time
- Data transfers automatically generated by OmpSs

Single point of change

```
#pragma omp target device(fpga) \
                implements(matrix_multiply) //num_instances(2)
#pragma omp task in(a,b) inout(c)
          void   matrix_multiply_fpga(float a[BS][BS],
                                      float b[BS][BS],
                                      float c[BS][BS]);
```

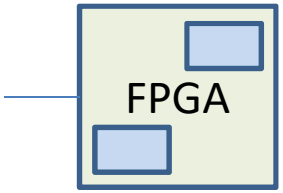




SMP

FPGA

```
# config file
ID   NUM  FUNCTION
0    2        matmul
```



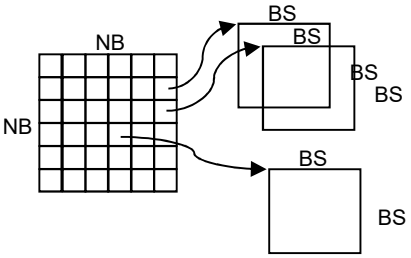
Heterogeneous programming

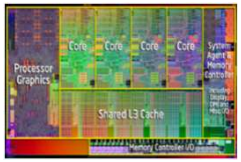
The “implements” approach

- Native C/C++ kernel!!! => Vivado HLS
- Kernel compiled “offline”... takes some time
- Data transfers automatically generated by OmpSs

Single point of change

```
#pragma omp target device(fpga) \
                implements(matrix_multiply) //num_instances(3)
#pragma omp task in(a,b) inout(c)
          void   matrix_multiply_fpga(float a[BS][BS],
                                      float b[BS][BS],
                                      float c[BS][BS]);
```

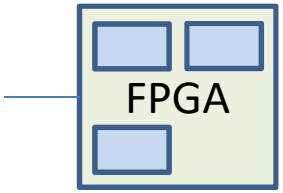




SMP

FPGA

```
# config file
ID   NUM  FUNCTION
0    3    matmul
```



Outline

Motivation

OmpSs

- compilation system (Mercurium autoVivado)
- code generation
- execution model (Nanos)
- instrumentation

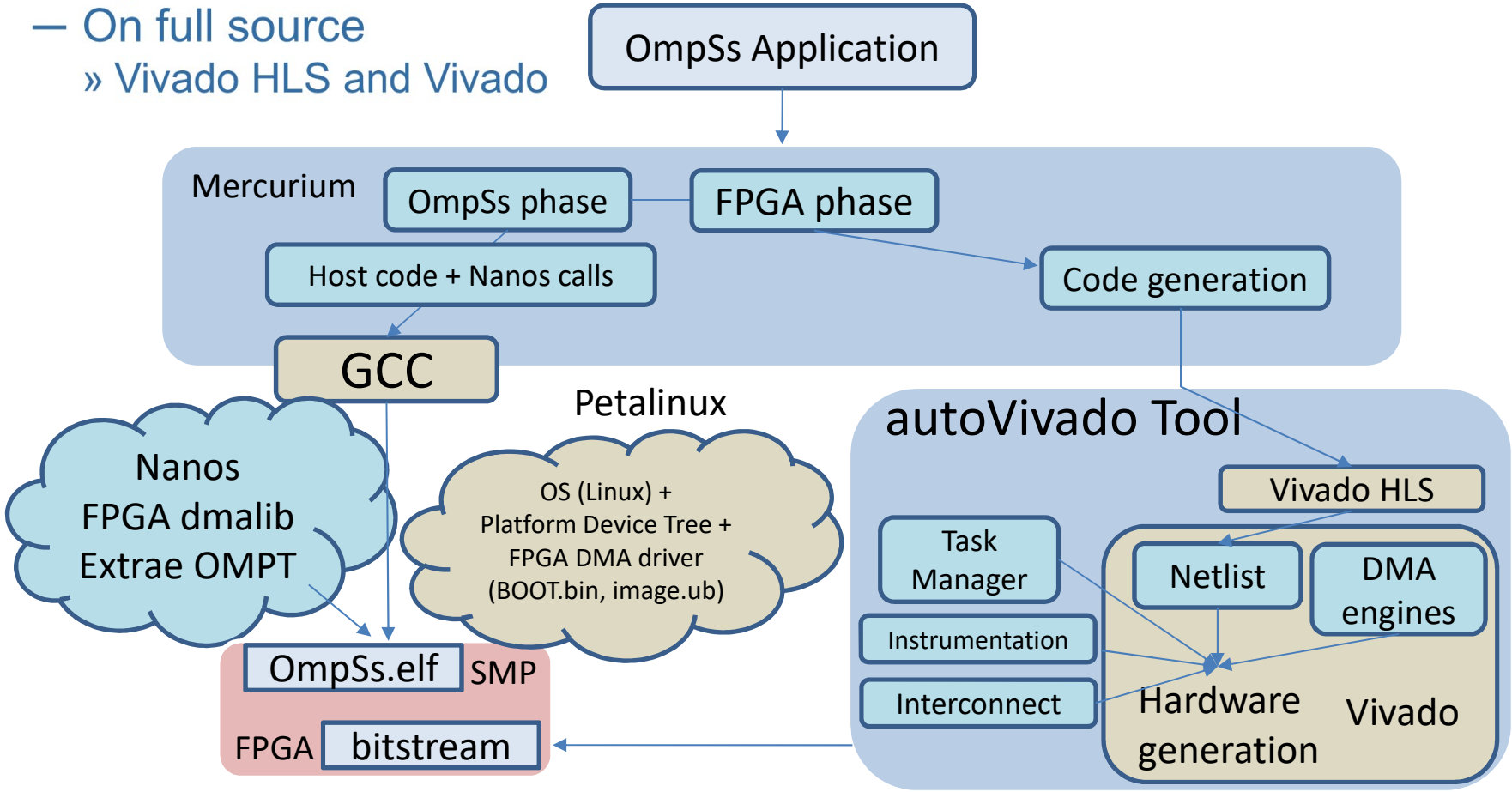
Evaluation

Conclusions & Future work

Mercurium, new FPGA toolchain: autoVivado

OmpSs transformations

– On full source
  » Vivado HLS and Vivado



OmpSs autoVivado

Coverage

```
#define BS 128

void matrix_multiply(float a[BS][BS], float b[BS][BS],float c[BS][BS])
{
#pragma HLS inline
  int const FACTOR = BS/2;
#pragma HLS array_partition variable=a block factor=FACTOR dim=2
#pragma HLS array_partition variable=b block factor=FACTOR dim=1
  // matrix multiplication of a A*B matrix
  for (int ia = 0; ia < BS; ++ia)
    for (int ib = 0; ib < BS; ++ib) {
#pragma HLS PIPELINE II=1
      float sum = 0;
      for (int id = 0; id < BS; ++id)
        sum += a[ia][id] * b[id][ib];
      c[ia][ib] += sum;
    }
}
```

Towards
Discrete FPGAs

Xilinx ZCU102
(work in progress)

Trenz Electronics Zynq U+
TE0808 XCZU9EG-ES1

SECO AXIOM Board
Zynq U+ XCZU9EG-ES2

Zynq-7000 Family
2x Cortex-A9 cores + FPGA
(32-bit platforms)

4x Cortex-A53 cores +
FPGA (64-bit platforms)

OmpSs autoVivado

Mercurium automatically generates wrapper

- Function accepting Vivado HLS standard arguments
  - » AXI bus connections with the PS
  - » Other management arguments

Wrapper and kernel source code are outlined to a separate file

autoVivado script is invoked to generate the bitstream

```
void matrix_multiply_hls_automatic_mcxx_wrapper(
                                        hls::stream<axiData> &inStream,
                                        hls::stream<axiData> &outStream,
                                        elem_t *mcxx_A,
                                        elem_t *mcxx_B,
                                        elem_t *mcxx_C_i,
                                        elem_t *mcxx_C_o)
{
   ...
```

OmpSs autoVivado

Wrapper function

- Sets the interfaces between PS and PL
- Declares BRAMs to store local data from in/out matrices

```
{
#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS INTERFACE axis port=inStream
#pragma HLS INTERFACE axis port=outStream
#pragma HLS INTERFACE m_axi port=mcxx_A
#pragma HLS INTERFACE m_axi port=mcxx_B
#pragma HLS INTERFACE m_axi port=mcxx_C_i
#pragma HLS INTERFACE m_axi port=mcxx_C_o
...
    elem_t  A[((bsize) - 1 - 0) + 1][256];
    elem_t  B[((bsize) - 1 - 0) + 1][256];
    elem_t  C[((bsize) - 1 - 0) + 1][256];
```

OmpSs autoVivado

Wrapper function

- Copies data from input arrays
- Executes kernel provided by the user
- Copies data back to output arrays

```
    …
    memcpy(A, (const elem_t *)(mcxx_A), (((bsize) - 1 - 0) + 1)*(256)*sizeof(elem_t ));
    memcpy(B, (const elem_t *)(mcxx_B), (((bsize) - 1 - 0) + 1)*(256)*sizeof(elem_t ));
    memcpy(C, (const elem_t *)(mcxx_C_i), (((bsize) - 1 - 0) + 1)*(256)*sizeof(elem_t ));

    matrix_multiply_fpga(A, B, C);

    memcpy( mcxx_C_o, (const elem_t  *)C, (((bsize) - 1 - 0) + 1)*(256)*sizeof(elem_t ));
    ...
}
```

Vivado HLS optimization directives

Vivado report on naive input code

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.75	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
46	26700	47	26701	none

Detail

Instance

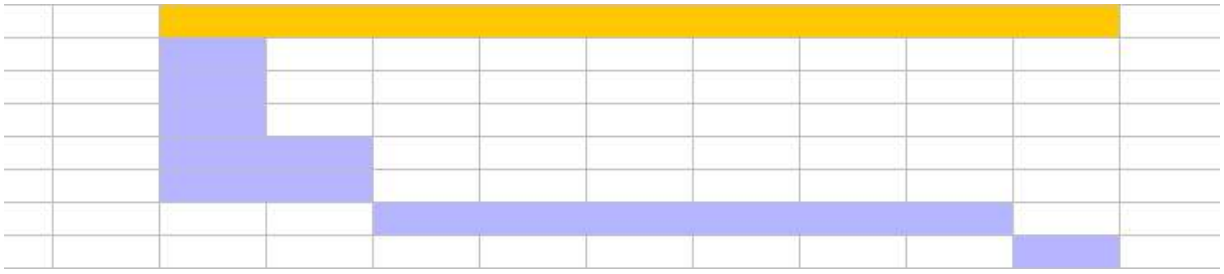
Loop

```
#pragma omp target device(fpga) copy_deps onto(0,1)
#pragma omp task in(vect_a[0:CONST_LITTLE_N-1], vect_b[0:CONST_LITTLE_N-1]) out(vect_c[0:CONST_LITTLE_N-1])
void vector_mult(int *vect_a, int *vect_b, int *vect_c)
{
   int i;

   for (i=0; i<CONST_LITTLE_N; i++)
   {
      vect_c[i]=vect_a[i]*vect_b[i];
   }
}
```

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	122
FIFO	-	-	-	-
Instance	16	4	2264	3128
Memory	12	-	0	0
Multiplexer	-	-	-	348
Register	-	-	973	-
Total	28	4	3237	3598
Available	280	220	106400	53200
Utilization (%)	10	1	3	6



Vivado HLS optimization directives

Vivado report on tuned code

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	6.08	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
73	73	73	73	none

Detail

Instance

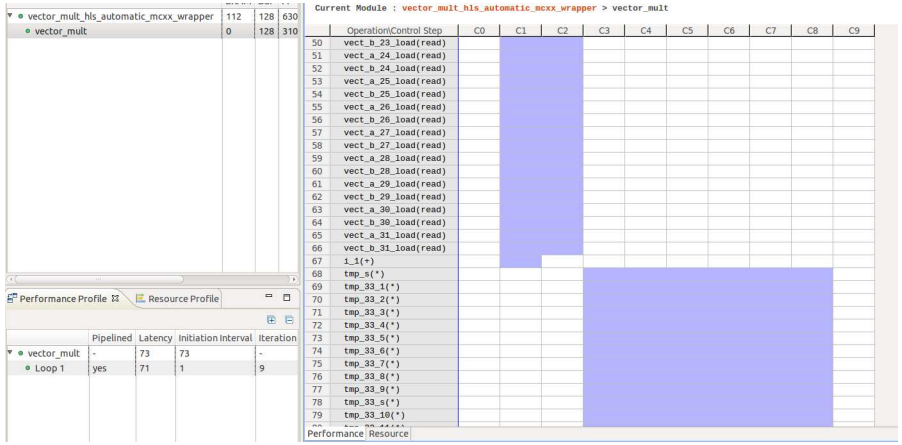
Loop

Utilization Estimates

Summary

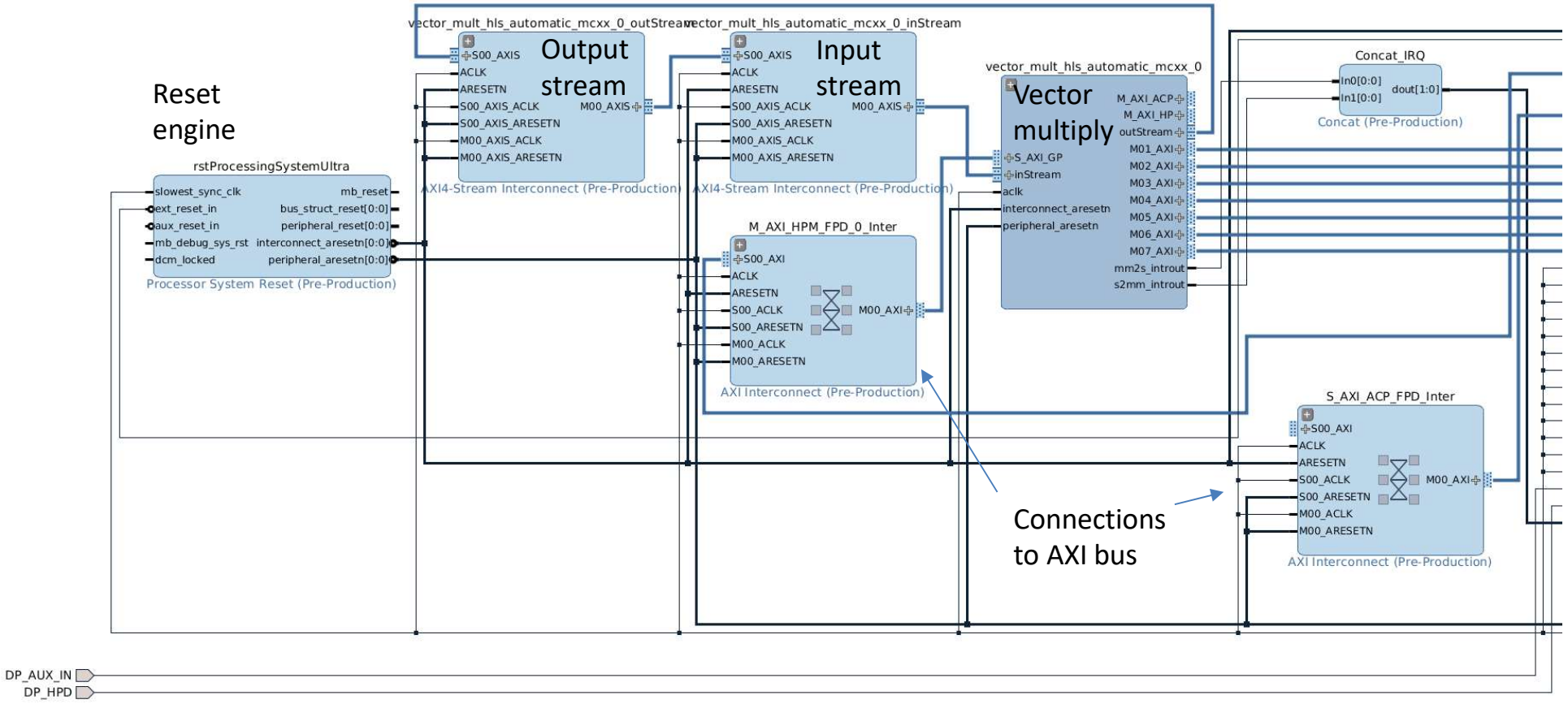
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	13
FIFO	-	-	-	-
Instance	-	128	0	0
Memory	-	-	-	-
Multiplexer	-	-	-	14
Register	-	-	3104	8
Total	0	128	3104	35
Available	280	220	106400	53200
Utilization (%)	0	58	2	~0

```
#pragma omp target device(fpga) copy_deps onto(0,1)
#pragma omp task in(vect_a[0:CONST_LITTLE_N-1], vect_b[0:CONST_LITTLE_N-1]) out(vect_c[0:CONST_LITTLE_N-1])
void vector_mult(int *vect_a, int *vect_b, int *vect_c)
{
   int i, ii;
#pragma HLS ARRAY_PARTITION variable=vect_a cyclic factor=CONST_BLOCK_FACTOR_MF
#pragma HLS ARRAY_PARTITION variable=vect_b  cyclic factor=CONST_BLOCK_FACTOR_MF
#pragma HLS ARRAY_PARTITION variable=vect_c   cyclic factor=CONST_BLOCK_FACTOR_MF

   for (i=0; i<CONST_LITTLE_N; i+=CONST_BLOCK_FACTOR_MF)
   {

   #pragma HLS PIPELINE II=1

// This will not hlep to create a pipeline=1
//     for (ii=i; ii<i+32; ii++)
//      vect_c[ii]=vect_a[ii]*vect_b[ii];
     for (ii=0; ii<CONST_BLOCK_FACTOR_MF; ii++)
      vect_c[i+ii]=vect_a[i+ii]*vect_b[i+ii];
   }
}
```



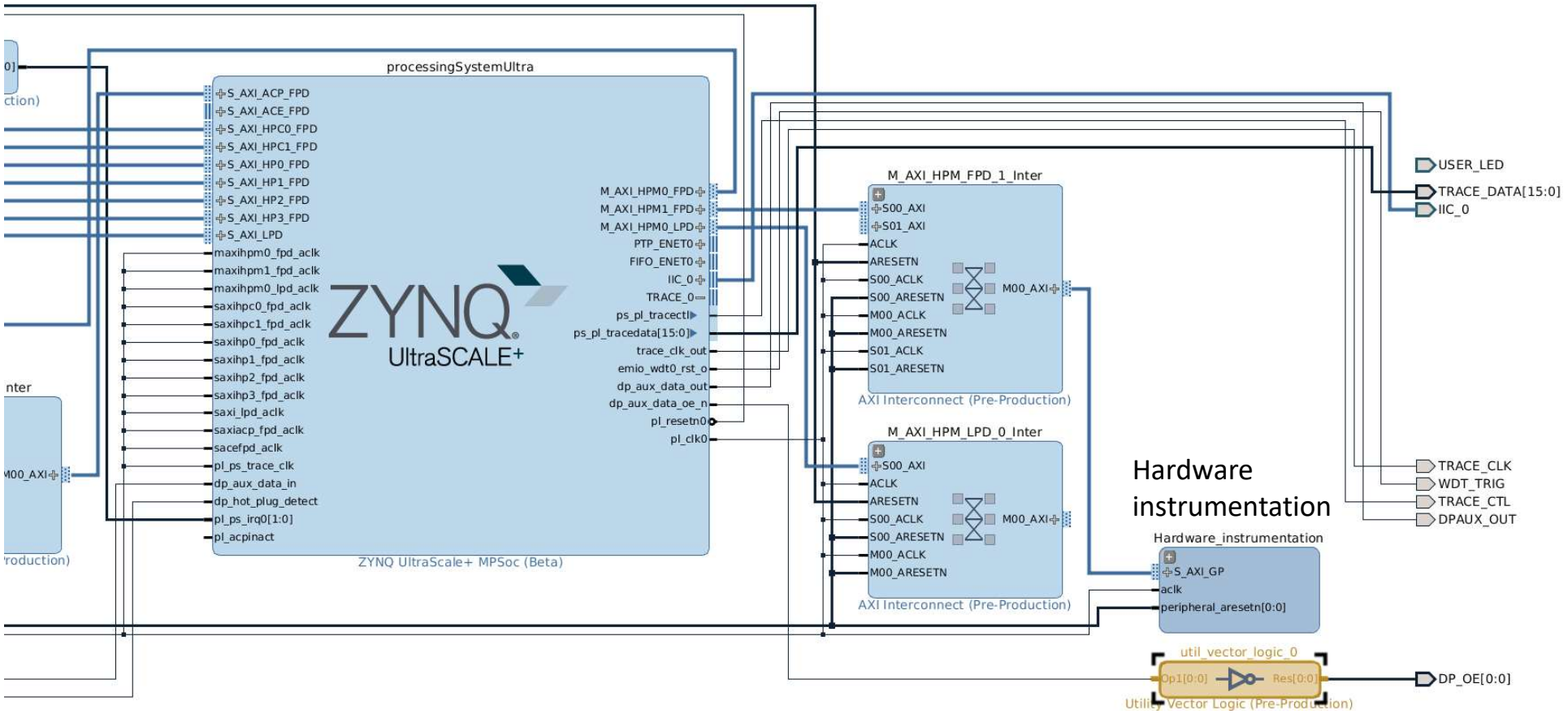
OmpSs autoVivado

Resulting synthesis from vector_multiply



OmpSs autoVivado

Resulting synthesis from vector_multiply



Outline

Motivation

OmpSs

- compilation system (Mercurium autoVivado)
- code generation
- execution model (Nanos)
- instrumentation

Evaluation

Conclusions & Future work

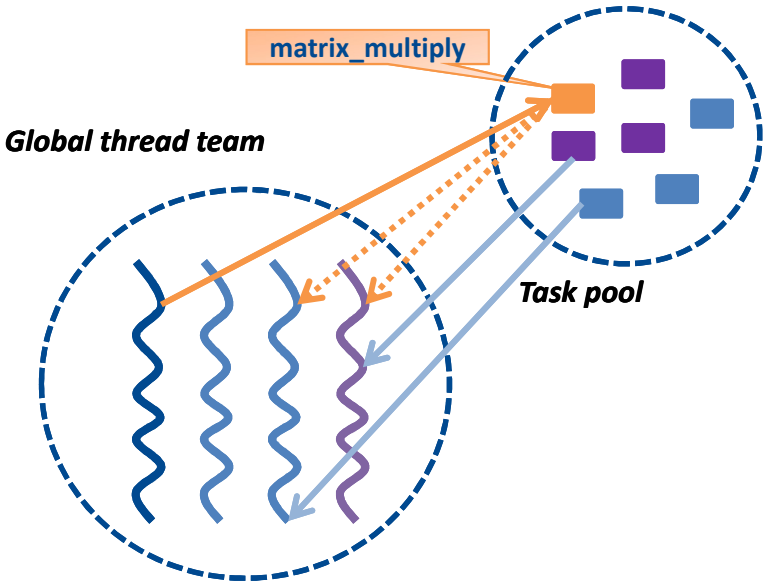
What happens on a “task”? Execution Model

Resources available

- OMP_NUM_THREADS / NX_GPUS / NX_NUM_FPGAS
- Master and workers execute SMP tasks
- **One representative** per (CUDA/OpenCL/FPGA) device/accelerator
  - » Experimenting with a single representative for N devices
    - *Good for FPGAs!!*

All get work from a task pool

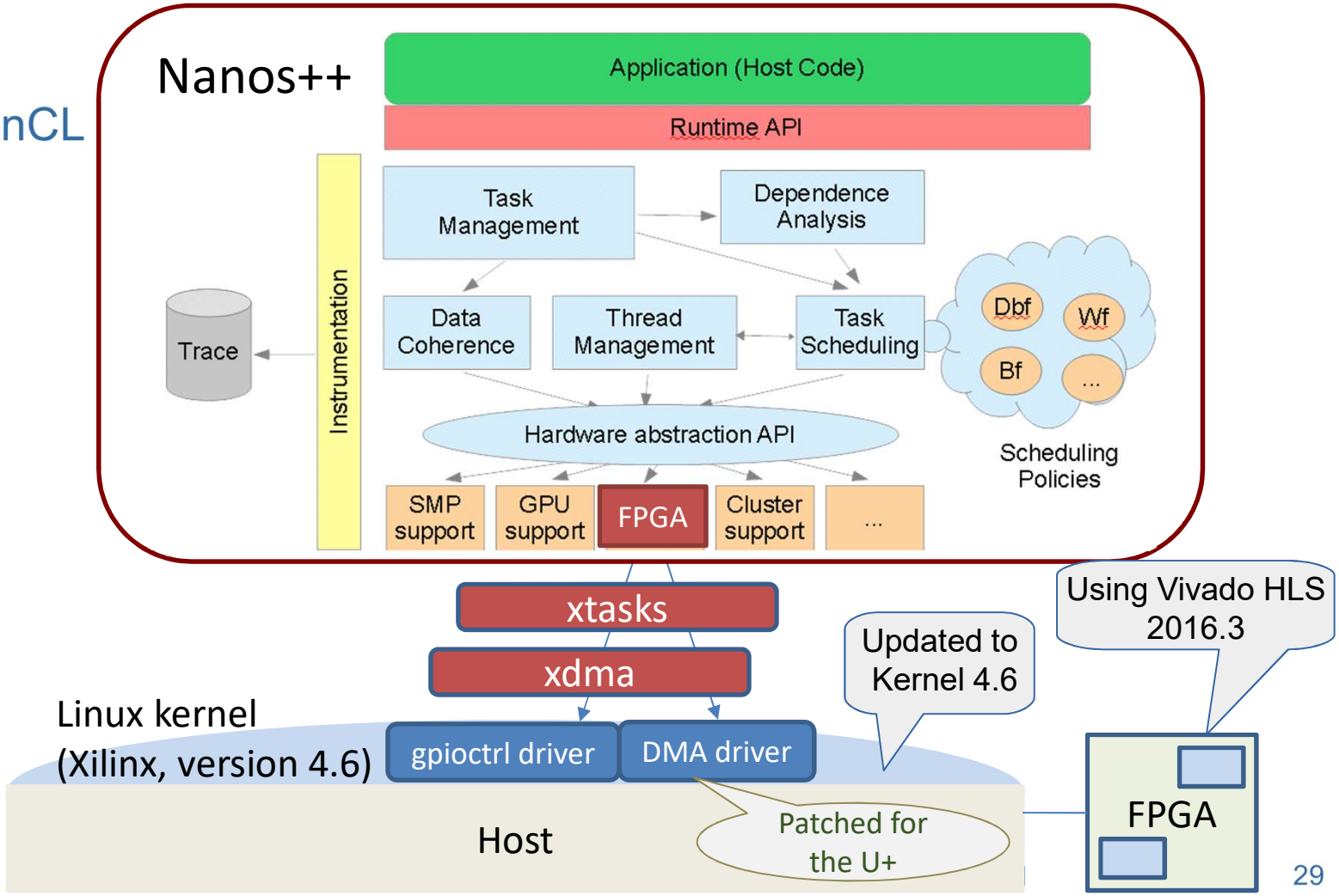
- And can generate new work
- Work is labeled with possible “targets”
- “implements” allow to exploit the parallelism on “all resources”



Nanos++ execution environment

Supports

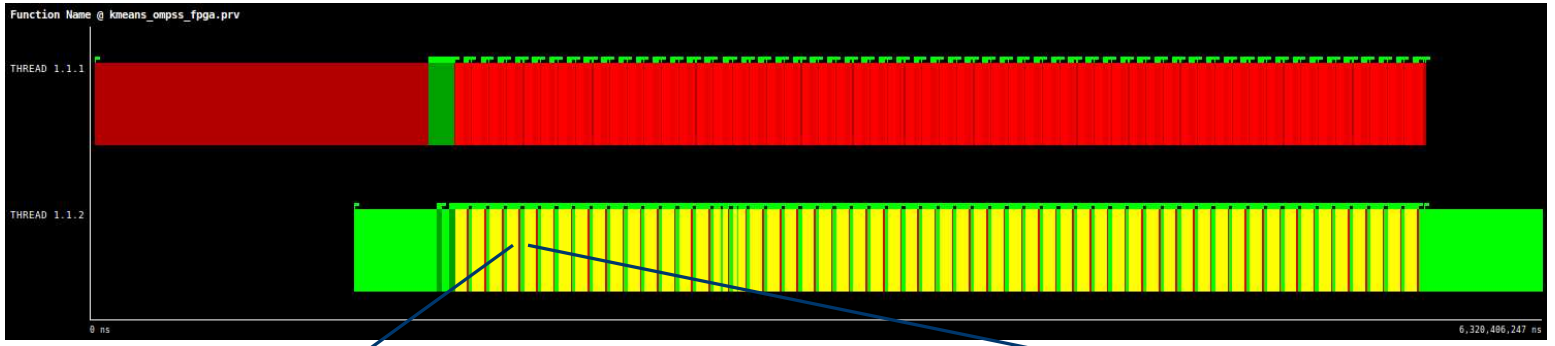
- SMP
- CUDA/OpenCL
- FPGAs
  - » Zynq 7000
  - » Zynq U+



Tracing the internals... of the FPGA!!

Master thread

FPGA representative

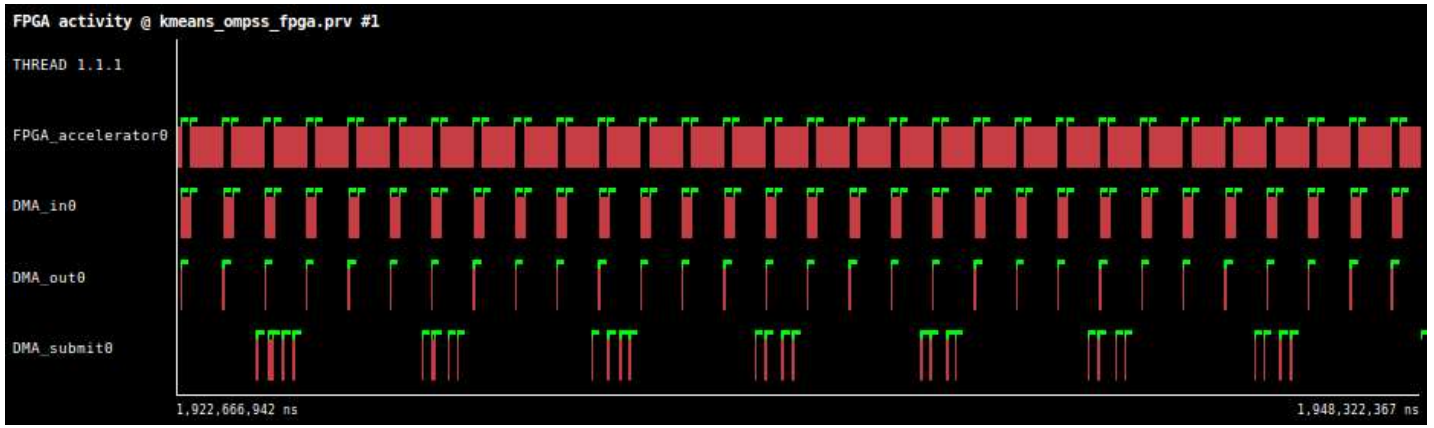


IP is active

Input channel

Output channel

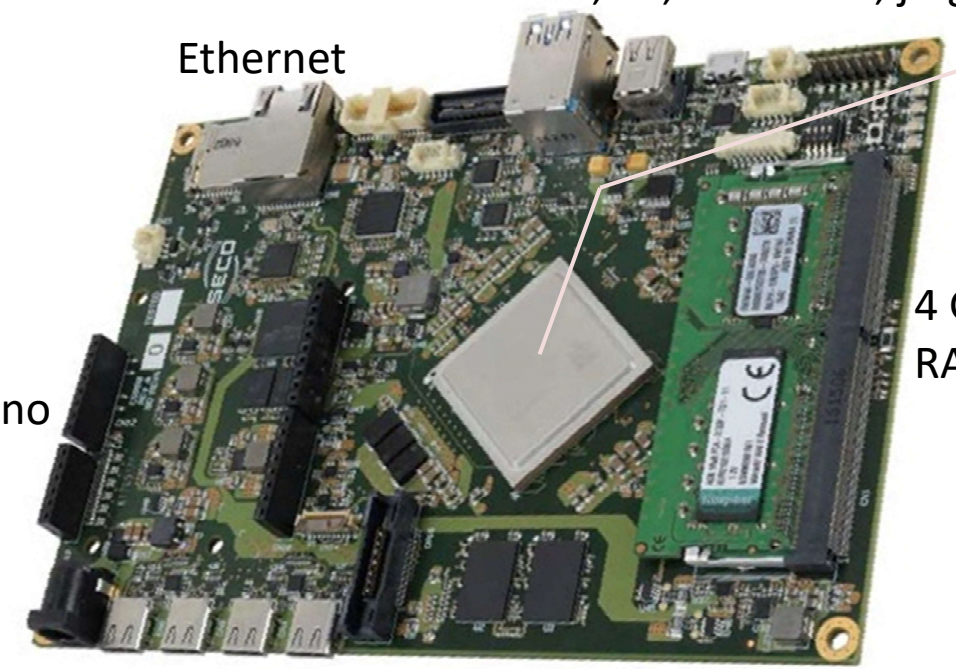
Submission



Performance evaluation

AXIOM board

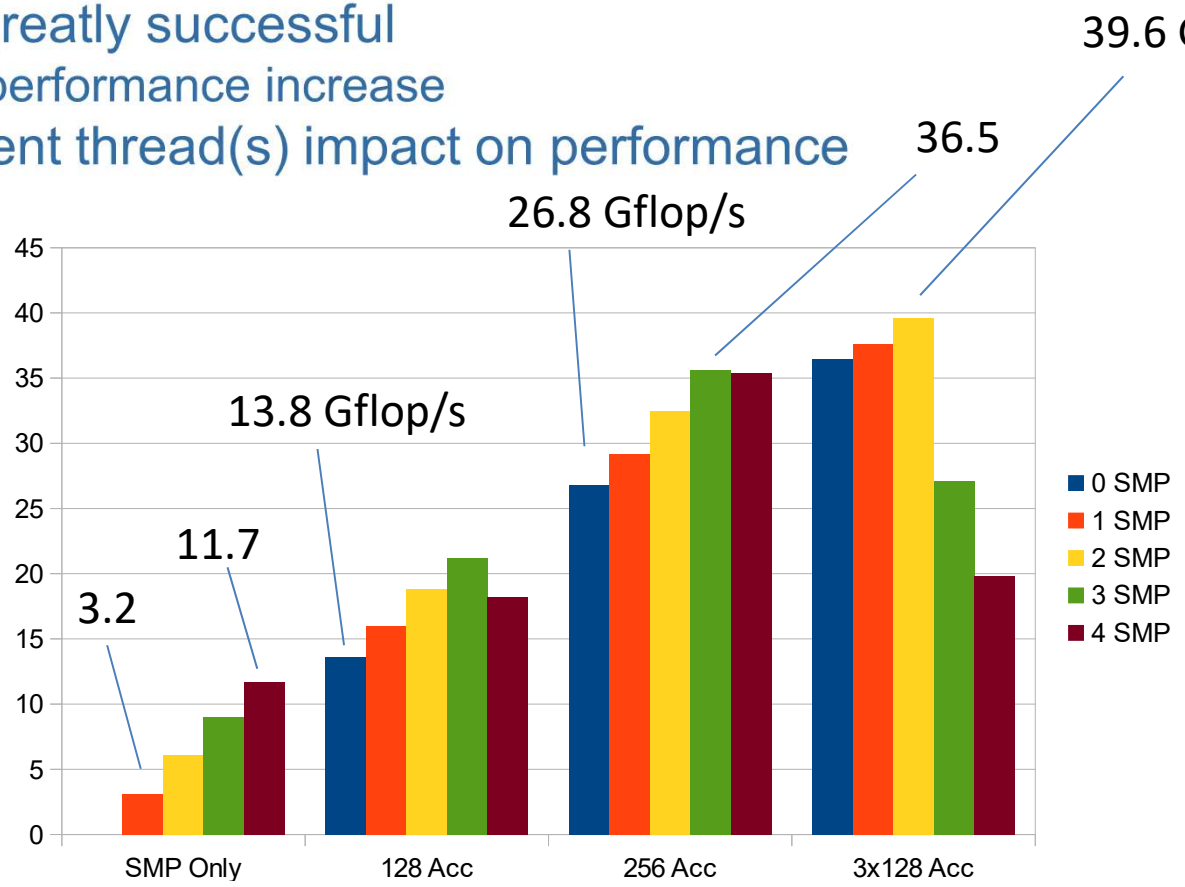
- Developed by SECO (Italy) in the AXIOM EU project
  - » Decision to use Zynq U+: May 2016
  - » First samples: March 2017



Zynq Ultrascale+ chip
- **ZU9EG FPGA device**
  - 600K Lcells
  - 548K FF
  - 274K LUTs
  - 4MB Block RAM
  - Up to 400MHz
- **Quad-core ARM A53**
  - 1.2GHz
- Dual-core ARM R5
- MALI GPU
  - (no OpenCL-capable)

Performance evaluation

- 1 IP matmul on the FPGA cleary outperforms 4 ARM cores
- "Implements" is greatly successful
  - » Adds up to 30% performance increase
- FPGA management thread(s) impact on performance
  - » Use Cortex R5?



OmpSs matrix multiply, single precision, 2048x2048, 300MHz on Ultrascale+


www.bsc.es
www.upc.edu
BSC
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación
UPC
UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH
Conclusions
3rd Int. Workshop on FPGA for HPC,
Tokyo, Japan - March 12th, 2018

Parallelization approach

Evolve from simple to more complex annotations

- Parallel loops “omp for”
- Parallel loops based on tasking “omp taskloop”
- Taskifying by hand “omp task”
- Adding support for heterogeneity “omp target device (...)”

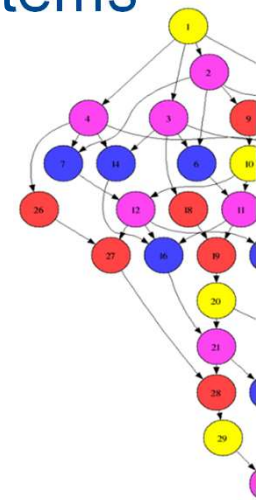
Check that the program works and provides correct results after each phase

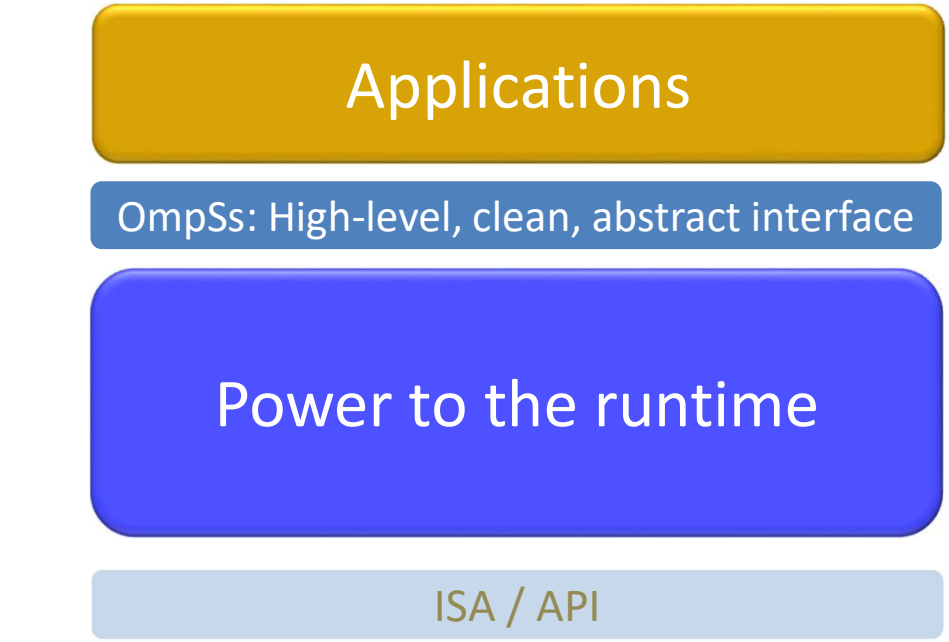
Check which alternative provides better performance and scalability

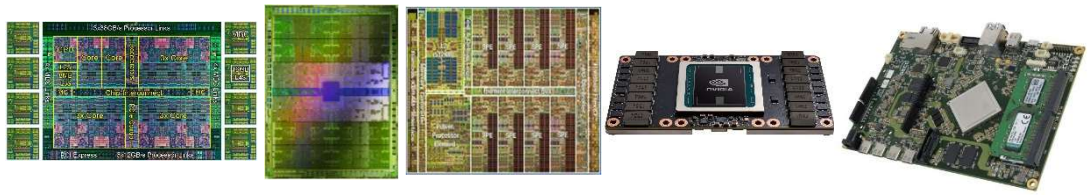
Conclusions

Providing higher level abstractions for heterogeneous systems

- SIMD code generation
- Nice integration of accelerators
  - » Memory management
  - » Automatic data transfers
  - » Automatic code generation
- Better decomposition of applications targeting exploitation of parallelism
  - » Coarse grain
  - » Tasking
  - » Task dependences
  - » Distant parallelism







Future work

Currently working on other algorithms on the FPGA

- From the AXIOM partners
  - » Face detection in high-definition video
  - » Voice analysis for smart-home
- K-means
- FFT

Analysis of the power consumption on the executions of those benchmarks

Plans to have the OmpSs@FPGA approach in the public domain

Please check http://www.axiom-project.eu

Acknowledgments

to the AXIOM project partners

to the BSC Programming Models team!!

Especial thanks to the FPGA team:

Daniel Jiménez-González

Carlos Alvarez

Antonio Filgueras

Miquel Vidal

Jaume Bosch...

Programming Models @BSC: https://pm.bsc.es

www.bsc.es

Using OmpSs to Run Applications in FPGAs

For further information, please contact

xavier.martorell@bsc.es

https://pm.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

3rd Int. Workshop on FPGA for HPC, Tokyo, Japan - March 12th, 2018

Additional slides

OmpSs: improving compiler technology

Example with Vivado HLS

```
#pragma omp target device(fpga) copy_deps onto(0,1)
#pragma omp task in(vect_a[0:CONST_LITTLE_N-1], vect_b[0:CONST_LITTLE_N-1]) out(vect_c[0:CONST_LITTLE_N-1])
void vector_mult(int *vect_a, int *vect_b, int *vect_c)
{
   int i;
#pragma HLS ARRAY_PARTITION variable=vect_a complete
#pragma HLS ARRAY_PARTITION variable=vect_b complete
#pragma HLS ARRAY_PARTITION variable=vect_c complete
   for (i=0; i<CONST_LITTLE_N; i++)
   {
   #pragma HLS PIPELINE II=1

      vect_c[i]=vect_a[i]*vect_b[i];
   }
}
```

Zynq-7000 Family
2x Cortex-A9 cores + FPGA
(32-bit platforms)

SECO AXIOM Board
Zynq U+ XCZU9EG-ES2

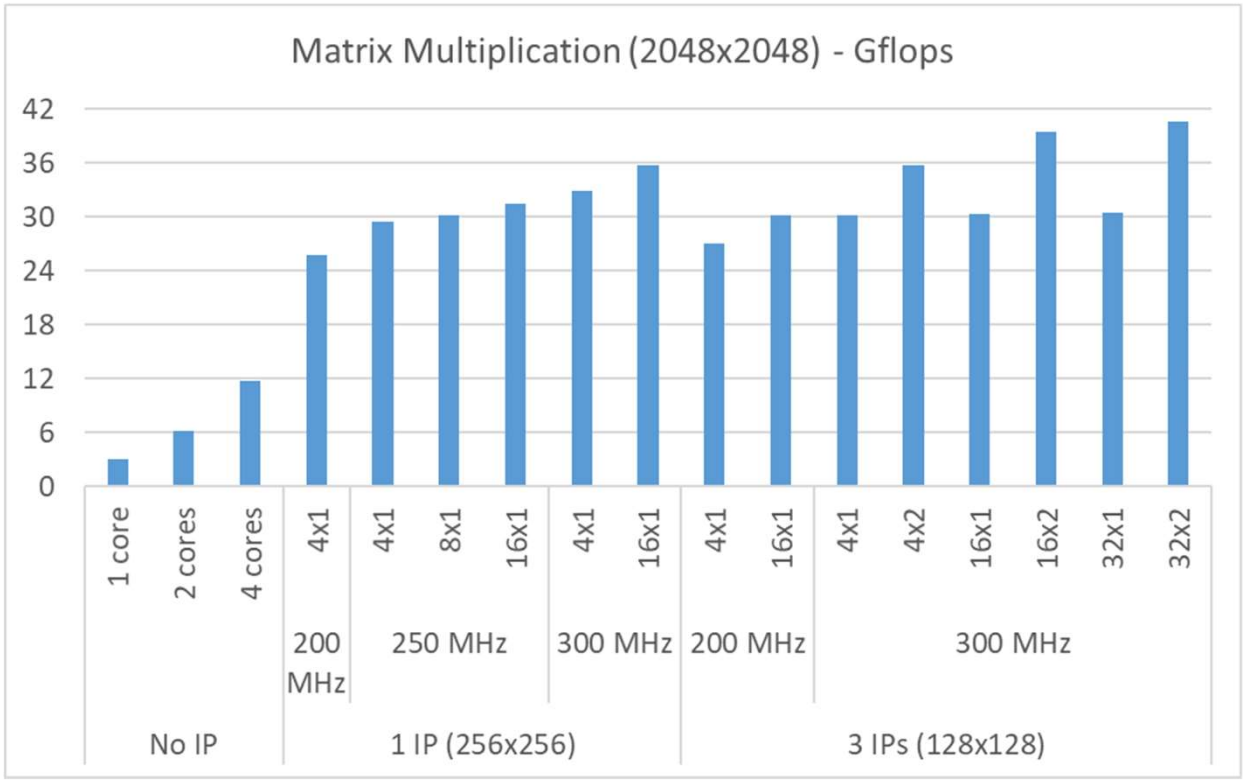
Towards
Discrete FPGAs

Trenz Electronics Zynq U+
TE0808 XCZU9EG-ES1

4x Cortex-A53 cores + FPGA (64-bit platforms)

Evaluation

Performance details



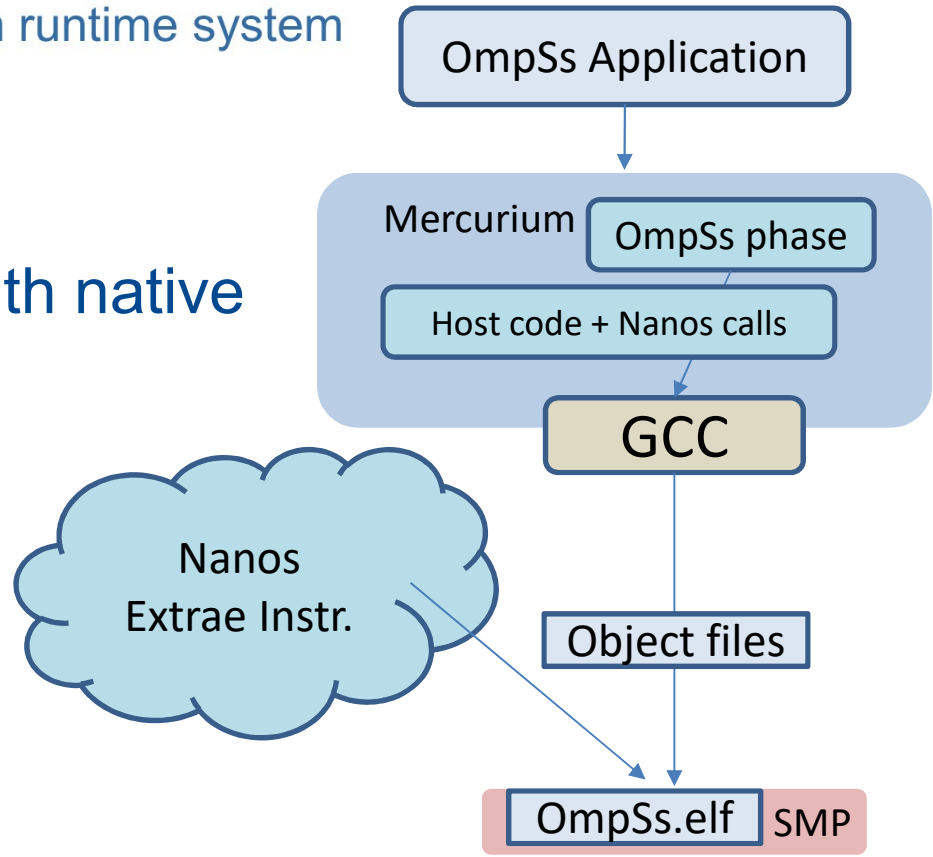
Mercurium compiler for SMP

Takes application annotated with OmpSs directives

- Inlined tasks
  - » Code offloading + invocation from runtime system
- Outlined tasks
  - » Invocation from runtime system

Compiles generated code with native compiler (gcc, llvm, icc...)

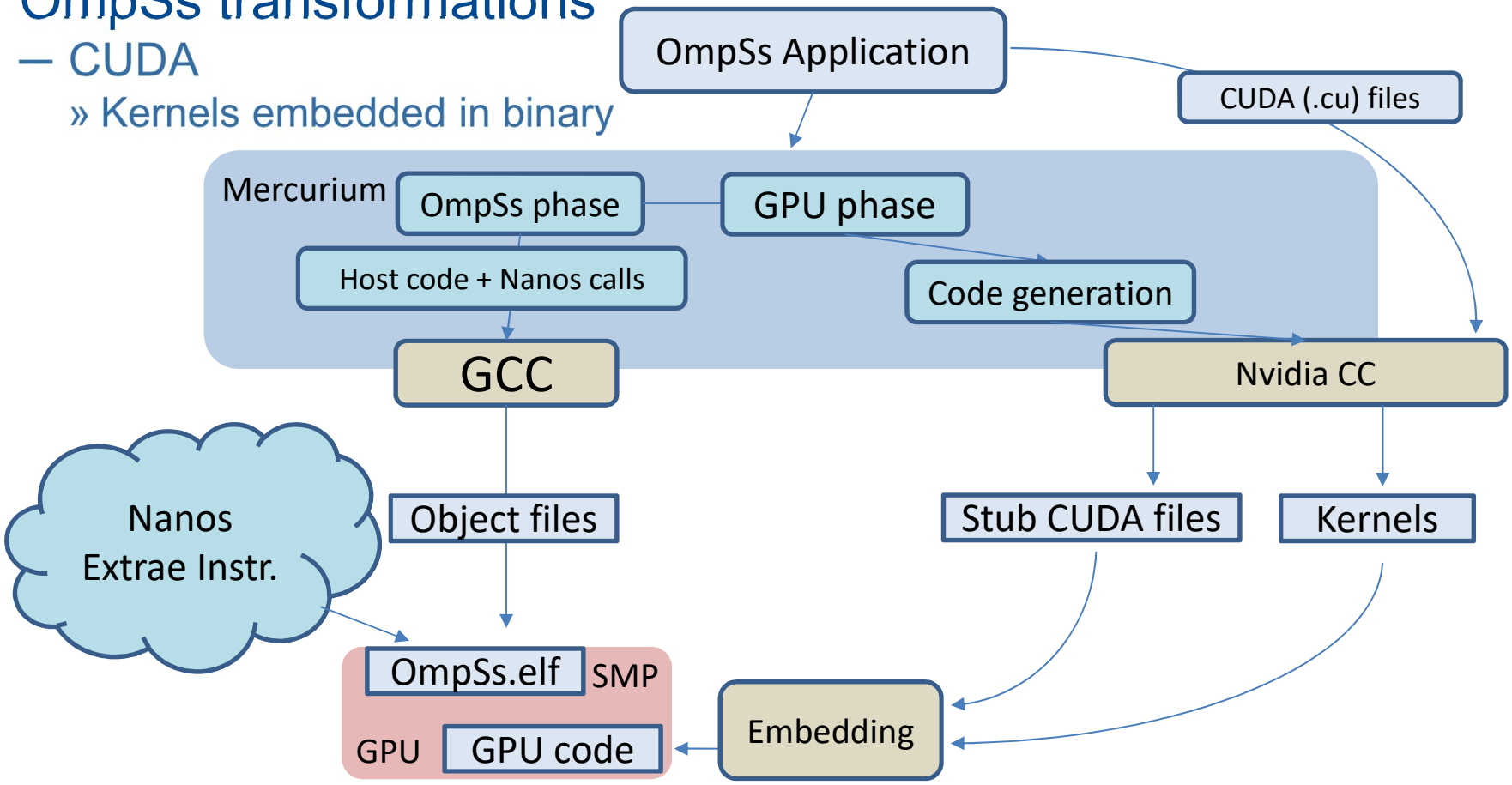
Links against Nanos and Extrae



Mercurium, support for CUDA

OmpSs transformations

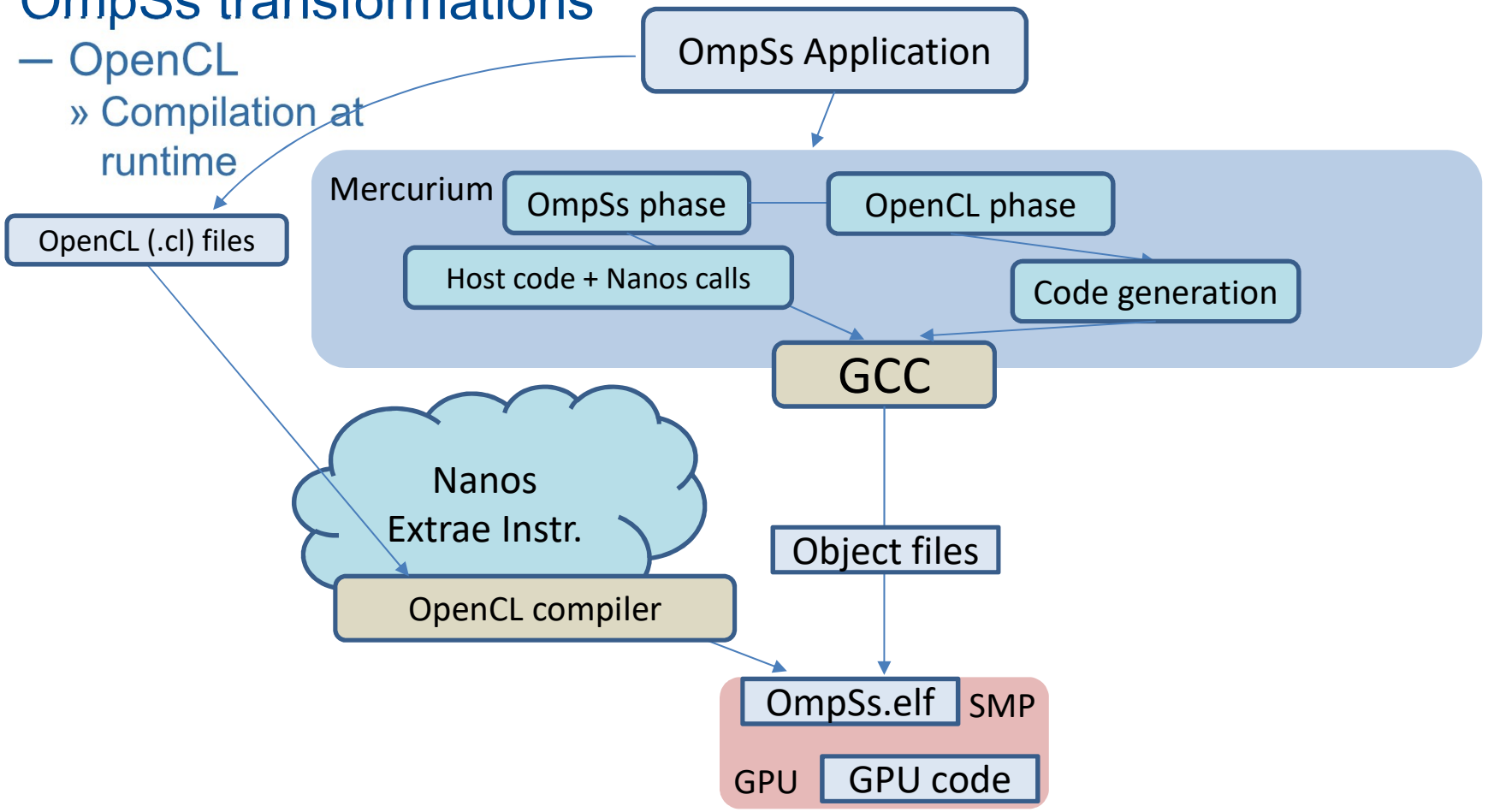
- CUDA
  - » Kernels embedded in binary



Mercurium, support for OpenCL

OmpSs transformations

- OpenCL
  - » Compilation at runtime



Execution environment

A few details on the steps to run the apps

- Environment
  - » Load (insmod) drivers: xdma.ko and gpioctrl.ko
  - » Download the bitstream
    - *Zynq 7000 family: cat program.bin >/dev/xdmacfg*
    - *Zynq U+: not supported yet (by Linux 4.6)*

- Run program: ./program <input arguments>
  - » Runtime application configuration
    - *program.xtasks.config file*
  - » With instrumentation
    - *export EXTRAE_CONFIG_FILE="extrae.xml"*
    - *LD_PRELOAD=libomptrace.so *
      *NX_ARGUMENTS="—instrumentation=ompt --fpga-freq=200" *
      ./program <input>

OmpSs autoVivado

How confortable is it?

- autoVivado is a collection of TCL scripts
  - » Currently driving VIVADO HLS 2016.3

- Given the compilation options...
  - » Board… axiom, zynq706, zynq702, zedboard
  - » Clock… e.g. 100ns
  - » Hardware instrumentation enabled
  - » Name of the Vivado HLS project to use
  - » Directory location for the Vivado HLS project

- ... the scripts get the proper hardware blocks from the Vivado library to build and link the configuration bitstream
- ... also generates a configuration file with the bitstream information

ID	NUM	FUNCTION
0	2	matmul
1	1	trsm
2	1	syrk

OmpSs autoVivado

Reports from compilation are available

— Kernel 256x256, 200MHz on Ultrascale+ xczu9eg

```
FF:       153654 used | 548160 available - 28.03% utilization
LUT:      111634 used | 274080 available - 40.73% utilization
BRAM:        648 used |   1824 available - 35.52% utilization
DSP:        1280 used |   2520 available - 50.79% utilization
2 kernel(s) synthesized. 144s elapsed.
Generating Vivado tcl script...
Vivado tcl script generated. 0s elapsed.
Starting Block Design generation...

****** Vivado v2016.3 (64-bit)
```

— Kernel 128x128, 100MHz

```
FF:       66642 used | 548160 available - 12.15% utilization
LUT:      51981 used | 274080 available - 18.96% utilization
BRAM:       305 used |   1824 available - 16.72% utilization
DSP:        640 used |   2520 available - 25.39% utilization
2 kernel(s) synthesized. 82s elapsed.
Generating Vivado tcl script...
Vivado tcl script generated. 0s elapsed.
Starting Block Design generation...
```