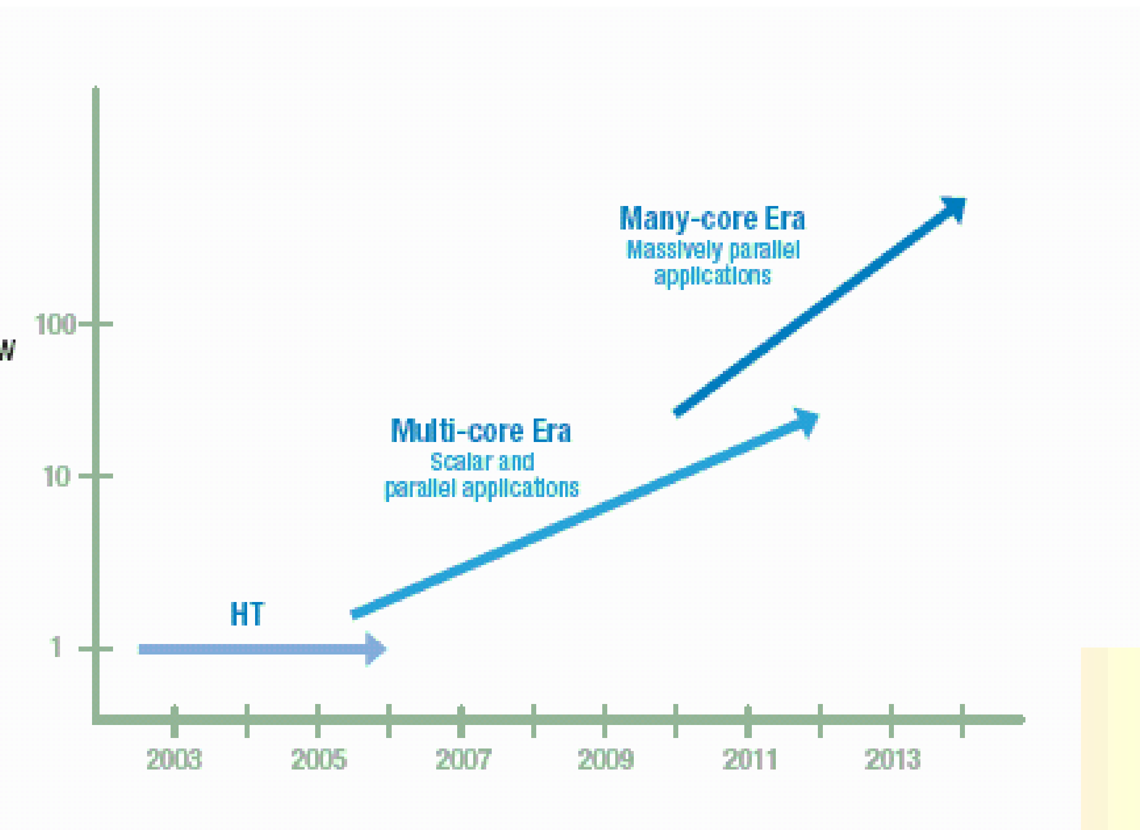
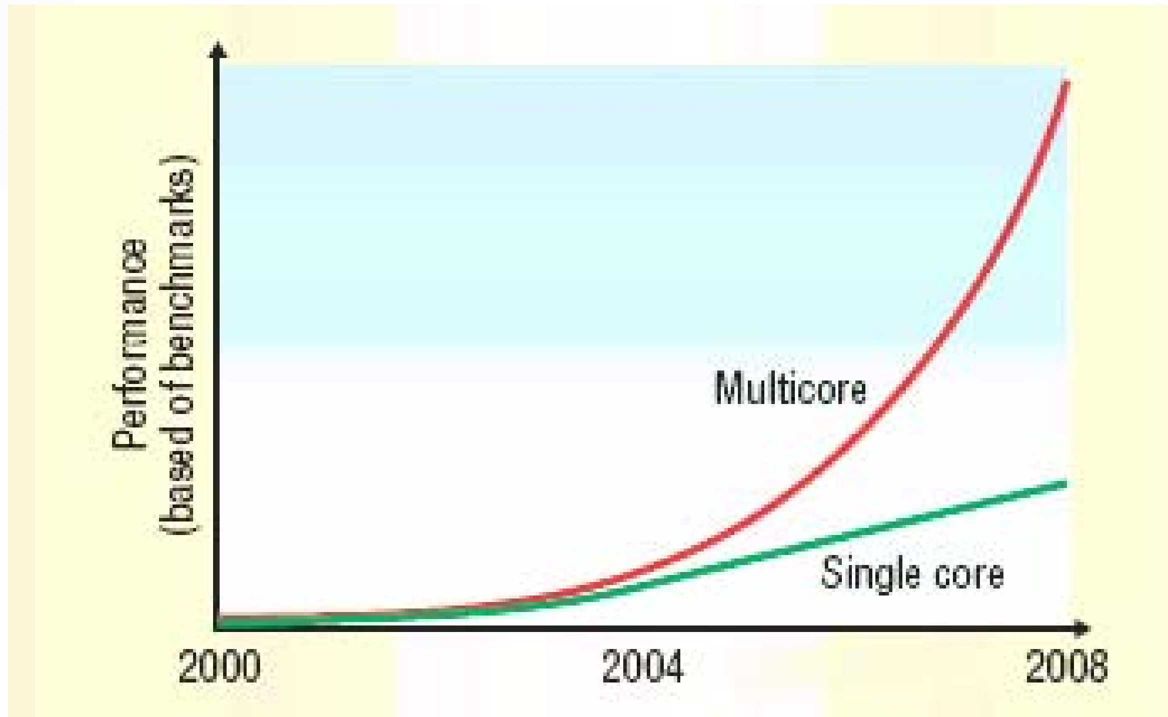
Extending the StarSs Programming Model to Platforms with Multiple Hardware Accelerators

Barcelona Supercomputing Center

Universidad Jaime I de Castellón





ource: Platform 2015: Intel® Processor and Platform Evolution for the Next

A few many-core architectures are already available:

A coarser-grain parallelism is present in platforms with multiple ccelerators:

- Distributed memory
- No hardware coherence mechanism
- Heterogeneity

Program these architectures as message-passing?

- Difficult!
- Rewrite existing libraries
- Likely not the most efficient solution

Extend the StarSs programming model to these platforms!

- "Sequential" program (i.e., single accelerator)
- StarSs run-time extracts coarse-grain parallelism to exploit multiple accelerators

StarSs tailored to multi-GPU platforms

Similarities between Cell B.E. and multi-GPU platforms:

- Cell B.E.: PPU + (8) SPUs with Local Store
- Tesla: CPU + (4) GPUs with Global Memory
- Interconnection buses: EIB vs. PCI Express

Many techniques from CellSs and SMPSs also valid for GPUSs

Minimal additions to the StarSs syntax

Possibility of hybrid GPU-CPU execution

Potentially valid for other multi-accelerator platforms or heterogeneous platforms

```
lobal__ void matmul( float * A, float * B, float * C ){

* CUDA Code */
```

```
 main( void ){
... */
 (i = 0; i < N; i++)
for (j = 0; j < N; j++)
   for (k = 0; k < N; k++)
       matmul ( A[i][k], B[k][j], C[i][j] );
..*/
```

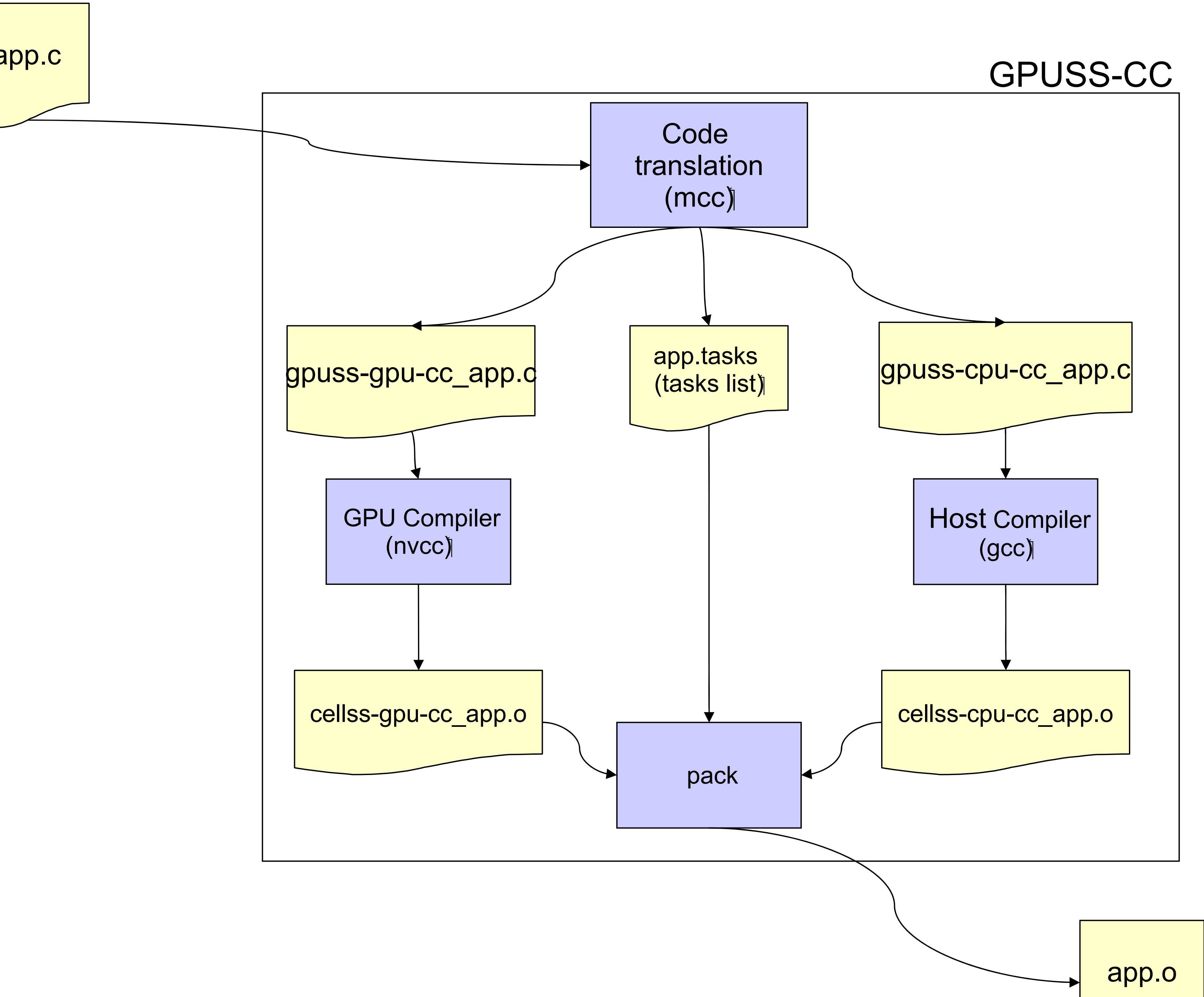
```
agma css task input(A[BS][BS], B[BS][BS]) inout(C[BS][BS]) device
A(dimGrid, dimBlock)
lobal__ void matmul( float * A, float * B, float * C ){

* CUDA Code */

 main( void ){
... */
 (i = 0; i < N; i++)
for (j = 0; j < N; j++)
   for (k = 0; k < N; k++)
       matmul ( A[i][k], B[k][j], C[i][j] );
..*/
```

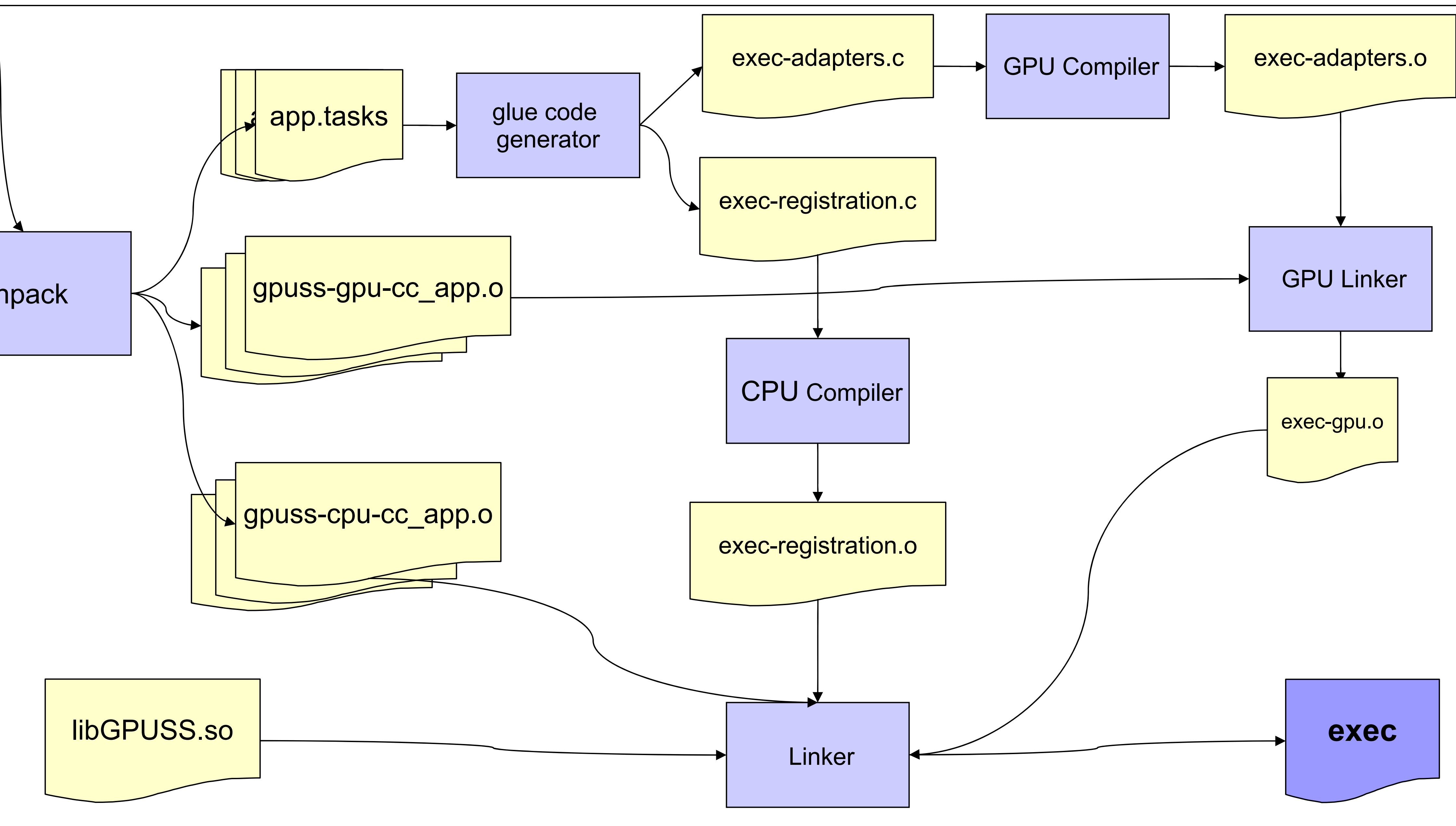
Task definition:
- Input/output parameter
- Type of accelerator
- Execution configuratio

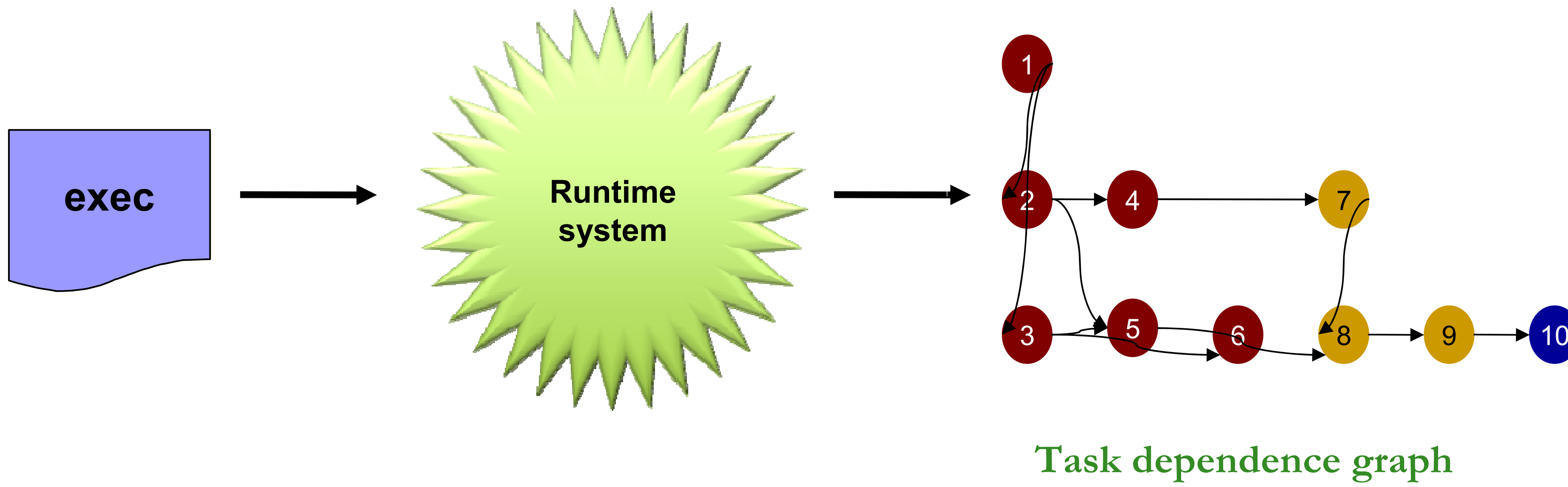
Main program:
- No explicit data transfers or allocation
- No explicit execution configuration


app.c
GPUSS-CC
Code translation (mcc)
gpuss-gpu-cc_app.c
app.tasks (tasks list)
gpuss-cpu-cc_app.c
GPU Compiler (nvcc)
Host Compiler (gcc)
cellss-gpu-cc_app.o
cellss-cpu-cc_app.o
pack
app.o

o

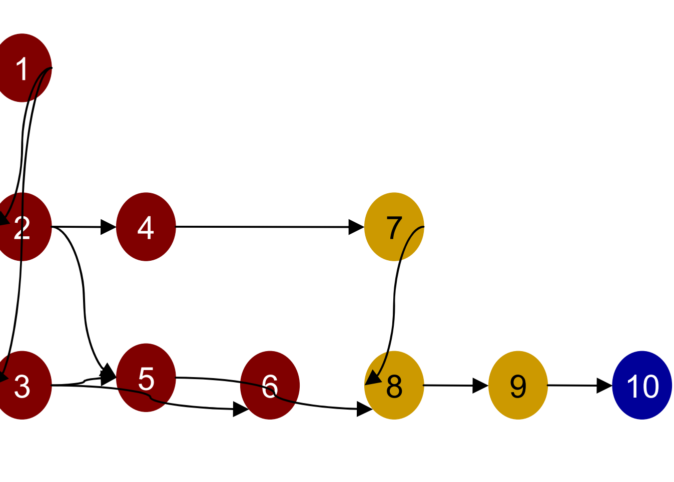
GPUSS





Detection of tasks and dependences at run-time

Generation of graph



Task dependence graph

Multi-GPU system
(Nvidia Tesla)

Scheduling of tasks

Mapping of tasks to GPUs or CPUs

Data transfers and device memory management

Exploitation of locality (data transfers reduction)

orry, no results or conclusions yet, this is ongoing work!

Extending the StarSs Programming Model to Platforms with Multiple Hardware Accelerators

Barcelona Supercomputing Center

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Universidad Jaime I de Castellón

UNIVERSITAT JAUME·I

Files

- app.c: User code, with GPUSs annotations
- cellss-gpu-cc_app.c: specific code generated for the gpu (tasks code written in CUDA)
- cellss-cpu-cc_app.c: specific code generated for the cpu (main program)
- app.tasks: list of annotated tasks

Compilation steps

- mcc: source to source compiler, based on the Mercurium compiler (BSC).
- GPU compiler: CUDA compiler (Nvidia NVCC)
- CPU compiler: Generic CPU compiler
- pack: Specific CellSs module that combines objects (BSC)

Files

- exec-adapters.c: code generated for each of the annotated tasks to uniform call them (“stubs”).
- exec-registration.c: code generated to register the annotated tasks

Linker steps

- unpack: unpacks objects
- glue code generator: from all the *.tasks files of an application generates a single “adapters” file and a single “registration” file per executable
- CPU and GPU compilers and linker