Teaching Parallel and Distributed Systems Programming for 4Th year Computer Science and Other Disciplines Students

Marcelo Arroyo

Universidad Nacional de Río Cuarto - Argentina

Teaching PDC

- Teaching parallel and distributed systems programming to:
  1. Undergraduate (4Th year) computer science students
  2. People coming from other disciplines (math, engineering, physics, biology, ...)
- Main problems:
  - Heterogeneity of knowledge
  - Different expectations

Computer science students

- 4Th year students
- Good background on:
  - Programming language paradigms
  - Data structures and algorithm design
  - Concurrency topics

Other students

- Basic skills on (imperative) programming
- Limited knowledge on data structures and algorithms
- No background on concurrency

Top-down approach

- Using high-level parallel patterns
  1. Introduce classical problems (matrix multiplication, sorting, searching, ...)
  2. Solve by patterns composition
  3. Focus on achieving high performance
  4. Study/analyze/extend/improve pattern internals (threads, MPI, OpenCL, OpenMP, ...)

Some patterns used

- `task-pool<expr,p,c>`: master-slave pattern
- `pipeline<p,c,`$e_1, e_2, \ldots, e_k$`>`
- `map<data,expr>`: apply `expr` to chunks of `data`
- `reduce<data,op>`: reduce `data` using `op`

Advantages

- Reduce the gap with sequential programming
- Focus on problem solving
- At first, hide implementation details
- Parallel programs = composition of components
- Students becomes enthusiastic from the very beginning

Tools

- Distributed data structures
- C++ templates: SkeTo, VecCL, pdt, ...
- Targets: threads, Openmp, MPI, OpenCL
- Patterns presented graphically (with contracts)

Students differentiation

- Computer science students develop new patterns or re-implement on new targets
- Other students focus on problem solving and achieving better performance

Results

- We were able to teach both: computer science and other discipline students
- All students developed complex parallel programs from start
- The concepts are taught incrementally: from high-level designs to low-level implementation details
- This approach appears to reduce the gap with sequential programming
- Students can compare different implementations of a pattern (on different targets)