A Stream-Computing Extension to OpenMP

Antoniu Pop and Albert Cohen

Centre de Recherche en Informatique, MINES ParisTech, France

INRIA Saclay and LRI, Paris-Sud 11 University, France

MINES ParisTech

INRIA

Why a Streaming Extension ?

Performance
- hide memory/communication latency
- bypass global memory with on-chip communication

Expressiveness
- fill the gap in the sharing clauses
  - data can be *shared* or *private*
  - data should be able to **flow**
- sometimes pipelining is the only way to go [1]

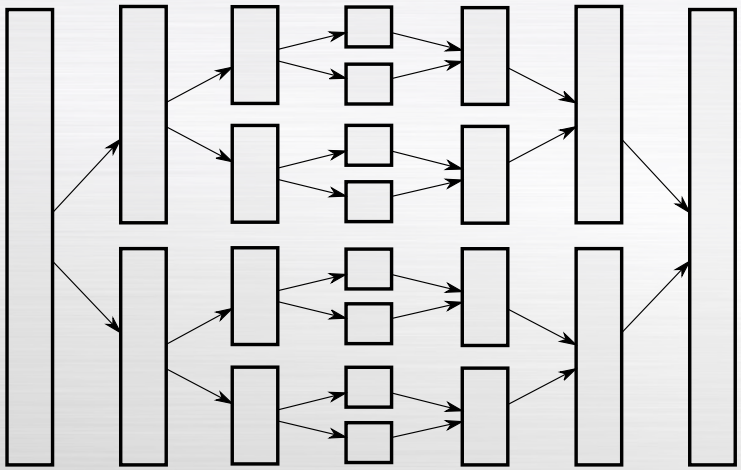
[1] V. Pankratius, A. Jannesari, and W. F. Tichy. Parallelizing Bzip2: A Case Study in Multicore Software Engineering. *IEEE Software*, 26(6):70–77, 2009.

Performance results

Opteron: 4-socket AMD quad-core Opteron 8380 (Shanghai) with 16 cores at 2.5GHz
Xeon: 4-socket Intel hexa-core Xeon E7450 (Dunnington) with 24 cores at 2.4GHz

FMradio: 12.6x speedup on Opteron and 18.8x on Xeon
802.11a: 13x speedup on Opteron and 14.9x on Xeon
FFT: 6.5x speedup on Opteron and 4.8x on Xeon

FFT data-flow graph

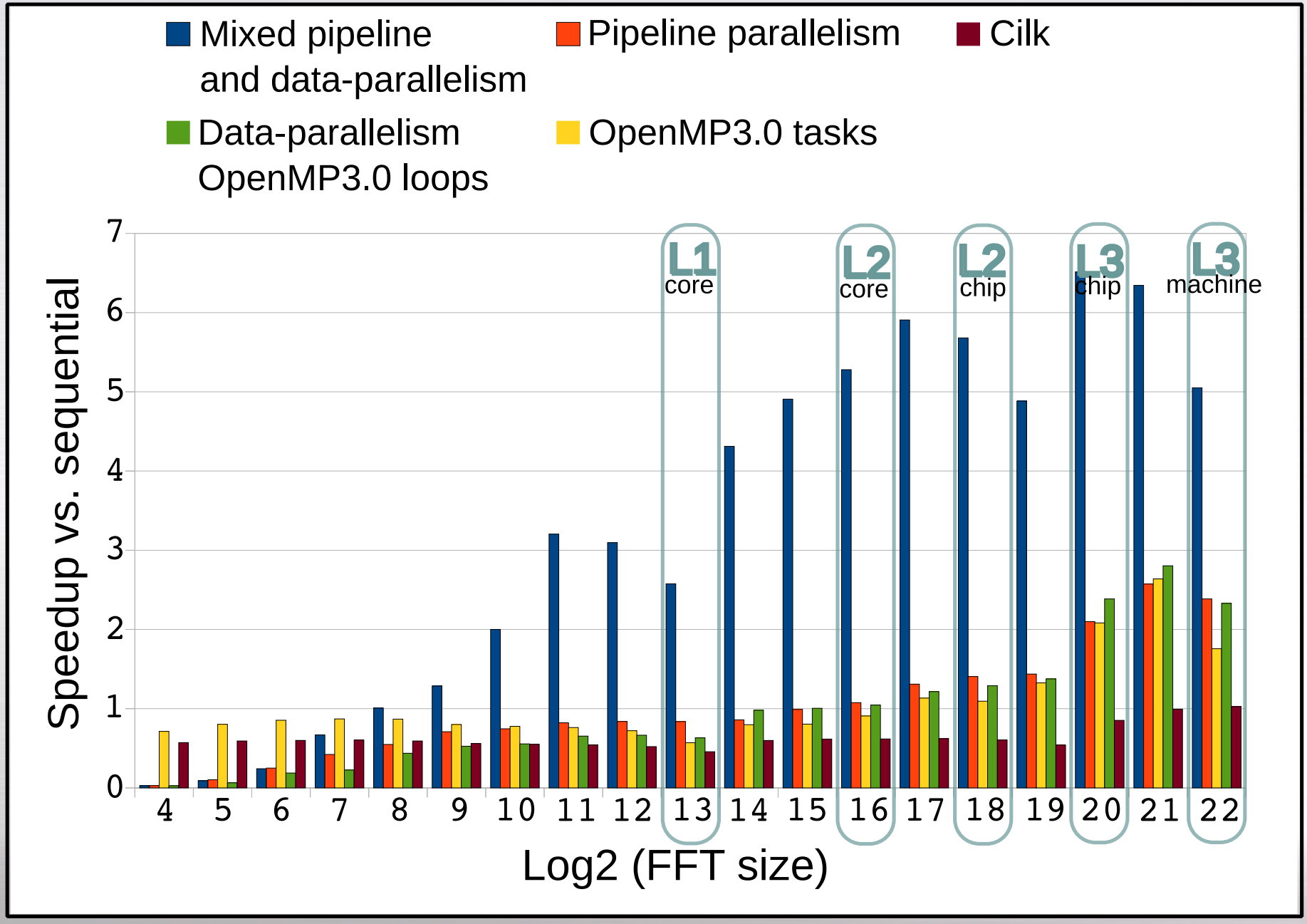


- Data-parallelism available in each stage (vertical sllice)

- Pipelining allows wavefront parallelization

- Maximum speedup (PRAM): roughly Log2 (FFT size) / 2

FFT performance on Opteron



A Simple yet Powerful Extension

Minor language extension
- only two sharing clauses: **input** and **output**
- simple and intuitive semantics

Seamless integration
- no impact on current semantics

Highly expressive
- as expressive as the StreamIt language

Implementation

- Public branch of GCC 4.5: **streamization**

For more details

- Poster
- Offline questions

For even more details

http://www.cri.ensmp.fr/classement/doc/A-416.pdf