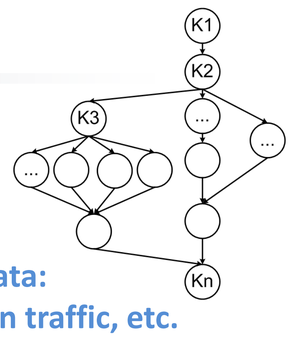
Kernel Partitioning of Streaming Applications: A Statistical Approach to an NP-complete Problem

Petar Radojković, Paul M. Carpenter,
Miquel Moretó, Alex Ramirez, Francisco J. Cazorla

Motivation

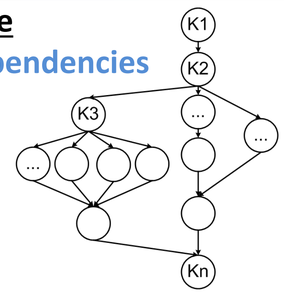
Stream programming languages

- **Programming of multithreaded applications is difficult**
- **A possible solution: Expose the parallelism to the compiler**
- **Stream programming languages (StreamIt, Brook, SPM)**
  - The application is presented as a stream graph
  - Suitable for applications that process long sequences of data: voice, image, multimedia, Internet and communication traffic, etc.



High pressure on the compiler

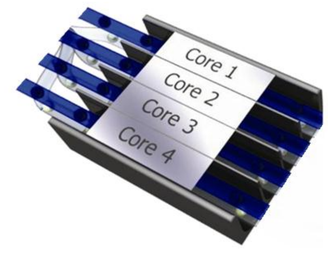
Source code
Explicit dependencies



Compiler

Complex code analysis and optimizations

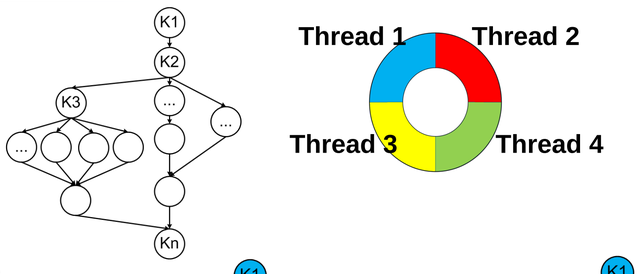
(Optimal) Multithreaded executable

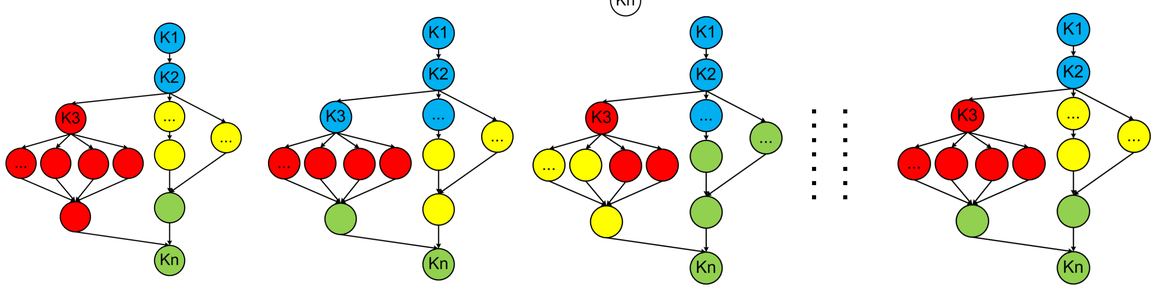


Problem

How to (optimally) partition kernels into software threads?

- **The compilation problem**
- **Color the nodes of the graph**





Kernel partitioning (KP) is an intractable problem

- **Vast exploration space (e.g. 10^{20} possible kernel partitions)**
- **NP-complete [Garey and Johnson, 1979]**

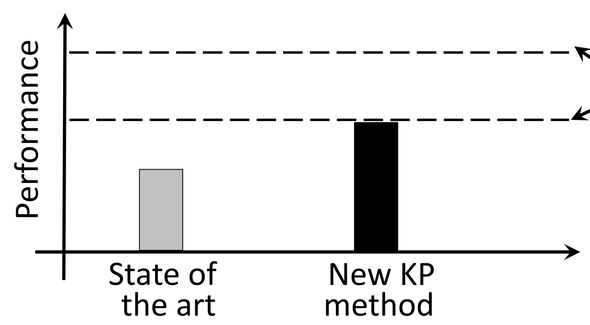
The importance of a good kernel partitioning

- **StreamIt 2.1.1 benchmark suite**
- **Exactly four software threads**
- **Observe the performance of good and bad KPs**
  - The performance difference ranges from **2.4x** to **3.9x**

The performance of the optimal kernel partitioning is unknown

State of the art approaches are based on heuristics

- **Try to find a good kernel partition**



Optimal kernel partition?

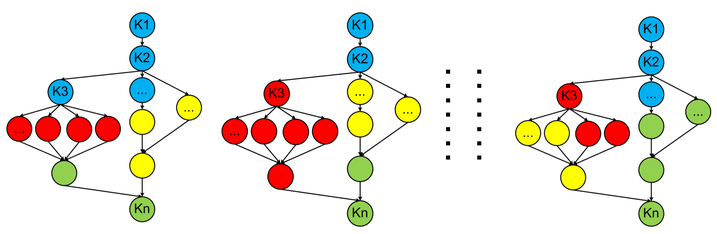
Should we keep working?

Are we close to the optimal?

Our proposal

Estimate the performance of the optimal kernel partition

Step 1: Execute random (i.i.d.) kernel partitions



Step 2: Measure the performance of each of them

15238	12659	.....	11988
12654	13564	.....	15684
16482	14551		10627

Step 3: Estimate the performance of the optimal partition

Statistical analysis
Extreme Value Theory

The optimal performance ranges from *X* to *Y*
[confidence level = 0.9; 0.95; 0.99]

Can random sampling find a good kernel partition (KP)?

13248	15728		11728
15468	12584		17588
24385	14658	⋮ ⋮	14385
12458	14458		10458
25847	09245		15847
12358	10444		09358
16548	15236		15628

Performance of 1000s of random kernel partitions

Do we capture a good one?

What is the probability to capture a good kernel partition in a random sample?

- **Total number of KPs: Finite but vast (e.g. 10^{20})**
- **Radnom sample of 1000 KPs**
  - Probability (capture the best KP) = 0
  - Probability (capture one out of the best 1% of KPs) = 99.99% *

* Radojković et al. *Optimal Task Assignment in Multithreaded Processors: A Statistical Approach.* In proceedings of ASPLOS 2012.

Results

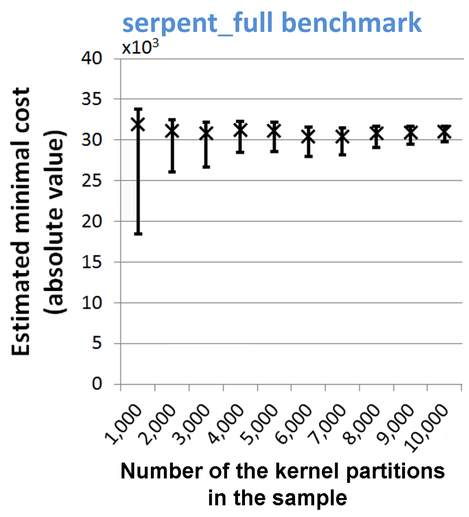
Can we apply EVT to the KP problem?

- StreamIt benchmark suite

Benchmark	Sampling method: Depth First Search	Edge Contraction	Edge Contraction with Filter	Uniformly Distributed
bitonic-sort	NA	✓	NA	✓
channelvocoder	✓	NA	NA	✓
des	NA	✓	✓	✓
fft	NA	✓	NA	✓
filterbank	NA	NA	NA	✓
.....	....	....	....	✓
serpent_full	NA	✓	NA	✓
vocoder	NA	✓	NA	✓

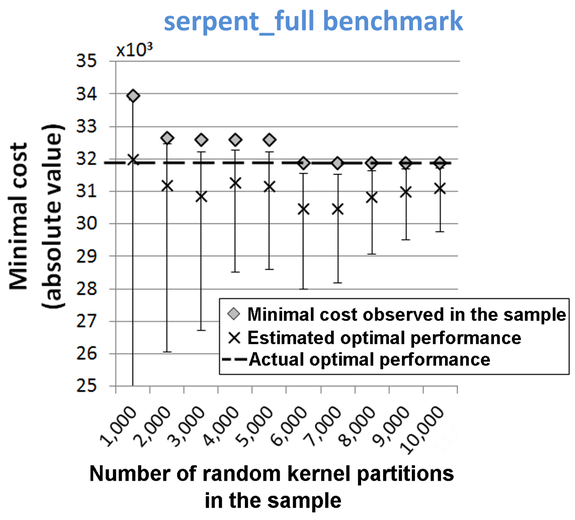
- **The samples should be uniformly distributed** ✓

Is the estimation precise?



- **Few 1000 KPs are sufficient for a precise estimation** ✓

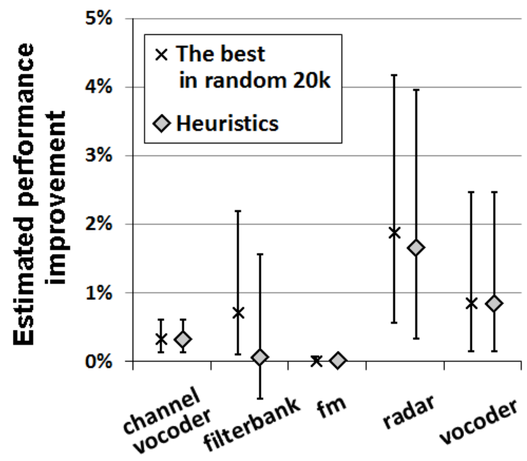
Is the estimation accurate?



- **The estimation is accurate** ✓

Can random sampling find a good KP?

- **Random sampling vs. Heuristics ****

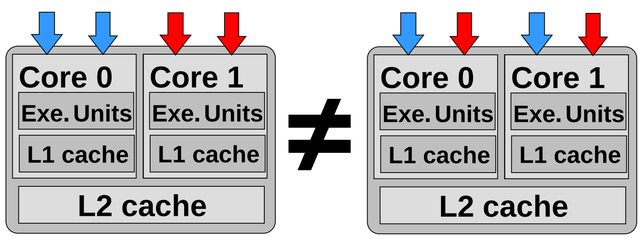


- **Random sampling provides very good results** ✓

** Carpenter et al. *Mapping Stream Programs onto Heterogeneous Multiprocessor Systems.* In proceedings of CASES 2009.

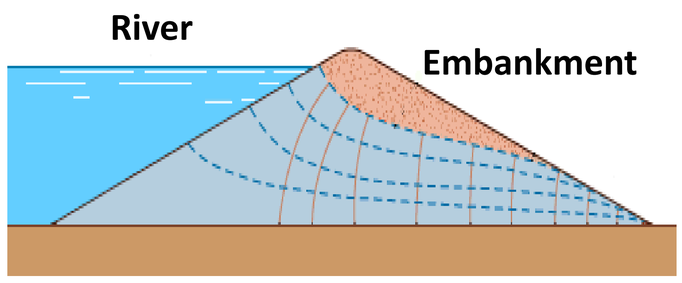
Application of Extreme Value Theory

Process scheduling for MT CPUs



Finance

Civil engineering



Numerous real-life problems