Communication Aware Actor Allocation of Stream Programs

Vitaly Nikolenko, S.M. Farhad, Bernhard Scholz

The University of Sydney

Outline

Stream Programming Paradigm

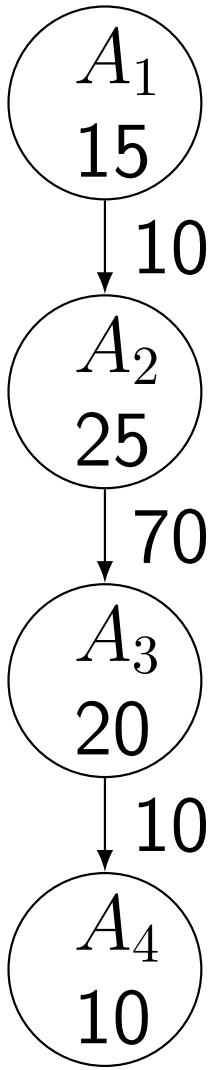
- Sequential programming languages:
  - Not well-suited for multi-core architectures
  - Insufficient parallel hardware abstraction
  - Programmer is responsible for identifying parallelism in programs
  - Tedious and error-prone
- Stream programming:
  - Natural and intuitive way to express parallelism

Stream Programming Paradigm (cont.)

- Application domains:
  - digital signal processing, audio, video, graphics and networking applications
- A computation is expressed by a stream graph:
  - actors/filters
  - data channels
- Examples of stream programming languages:
  - StreamIt, Brook, StreamFlex, openCL

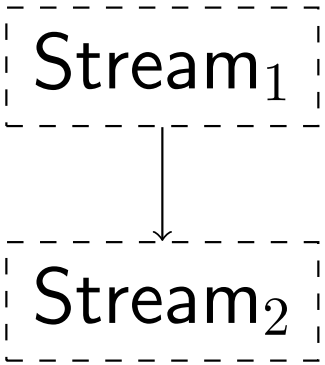
StreamIt Example

```
void->void pipeline Program() {
  add A1();
  add A2();
  add A3();
  add A4();
}
void->int filter A1() {
  int x;
  init {x=0;}
  work push 1 {push (x++);}
}
int->int filter A2() {
  work push 6 pop 1 {
      pop();
      // do some work
      push(); // 6 times
  }
}
int->int filter A3() {
  work push 1 pop 6 {
      pop(); // 6 times
      // do some work
      push();
  }
}
int->void filter A4() {
  work pop 1 {print(pop());}
}
```

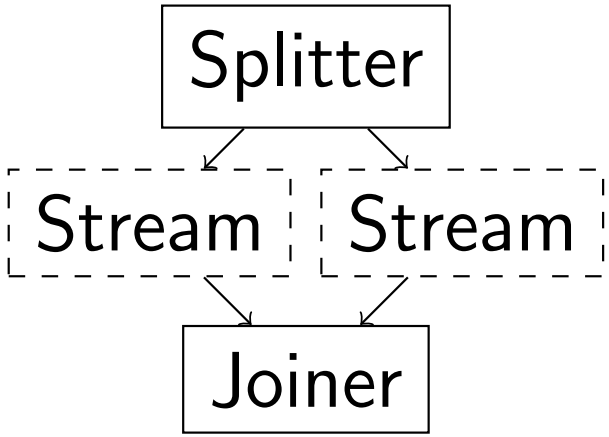


StreamIt structured stream graphs

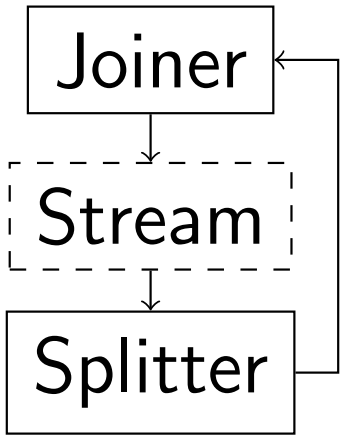
(a) Filter



(b) Pipeline



(c) Split-join



(d) Feedback loop

Stream Programming Paradigm (cont.)

The problem

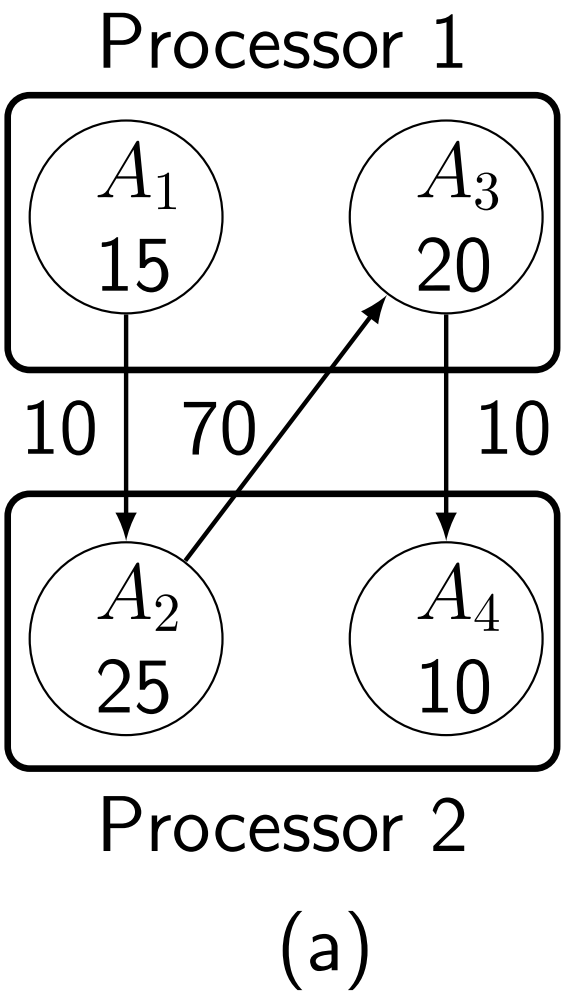
Mapping stream graphs onto processor cores:

- Minimize the total execution time (makespan):
  - Actor execution times + communication costs

Current research:

- Heuristics [1]
- No efficient approximation algorithms
- ILP models [2, 4, 3]

Communication Aware Allocation



Actor Placement Problem (APP)

Given a stream graph $G(V, E)$, a placement is a partiton of V into disjoint sets $S_1, ..., S_k$ where set S_i denotes the actors placed on processor core i. The runtime r_i is given by:

$$r_i = t(S_i) + \frac{1}{2}c(S_i, V \setminus S_i)$$

Objective: $\min_{S_i} \max_i r_i$

Contributions

- Prove APP NP-hardness
  - reduction from 3-partition
- Present $\log n$-approx for APP
- Demonstrate *instance* bounds that provide better bounds than $\log n$
- Evaluate our algorithm using StreamIt benchmark suite

Approximation Algorithm

- $\log n$-approximation (n is the number of actors)
- Greedy algorithm that uses dynamic programming as a subroutine
- Polynomial runtime $\mathcal{O}(np^3 \log n)$

Approximation Algorithm

Dynamic program as a subroutine (*PackP*):

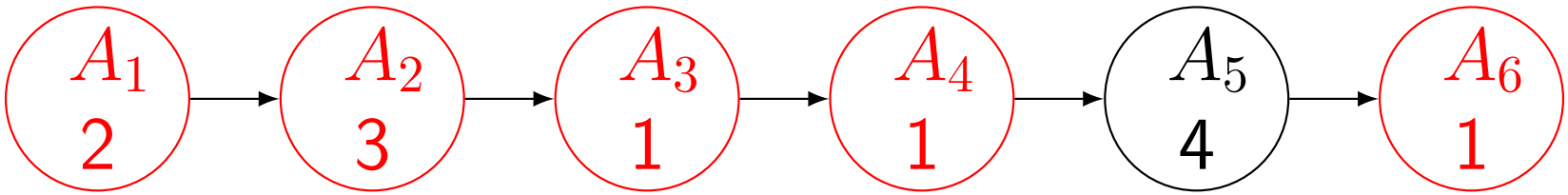
- Maximise number of actors placed on a processor
- Subject to the makespan constraint

Greedy component (*Oracle*):

- "Greedily" assign unallocated actors
- Reassign actors based on their execution and communication costs

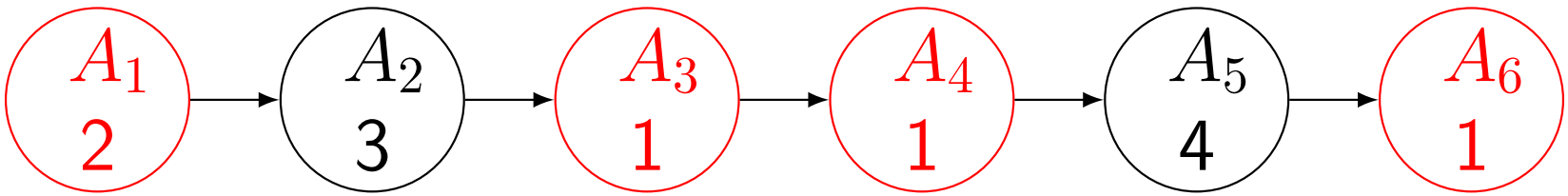
Example - PackP

- Let makespan $\Pi = 10$
- 3 processor cores
- Set of unassigned actors
  $U = \{A_1, A_2, A_3, A_3, A_4, A_5, A_6\}$



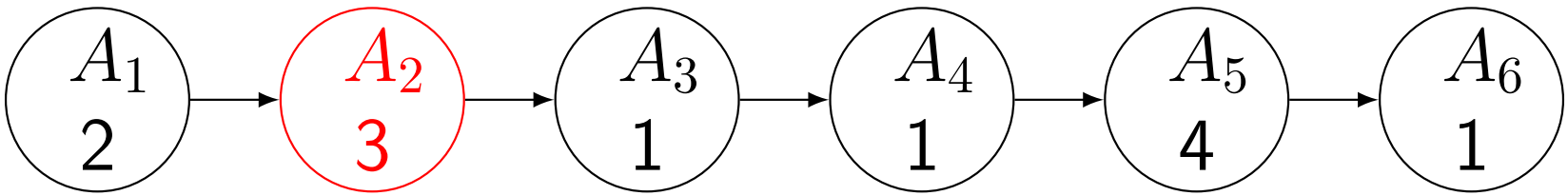
Example - PackP

- Let makespan $\Pi = 5$
- 3 processor cores
- Set of unassigned actors
  $U = \{A_1, A_2, A_3, A_3, A_4, A_5, A_6\}$



Example - PackP

- Let makespan $\Pi = 5$
- 3 processor cores
- Set of unassigned actors $U = \{A_2, A_5\}$



Evaluation

Instance Bounds

Approximation ratio of a specific benchmark instance:

$$LR \leq OPT \leq APX$$

Instance bound $\frac{APX}{LR}$ is an upper bound on the approximation ratio of an instance.

Evaluation - 2 cores

			2 cores		
Benchmark	n	$\|E\|$	k	$\frac{APX}{LR}$	$\frac{APX}{OPT}$
MergeSort	31	37	2	1.52	1.368
ChanVocoder7	55	70	3	1.244	1.225
MatrixMult	52	88	2	1.579	1.477
FFT2	26	26	2	1.15	1.136
FMRadio	67	85	3	1.393	N/A
BeamFormer	34	39	4	1.055	1.038
DCT	22	28	3	1.32	1.269
TDE	55	56	3	1.207	1.207
RadixSort	13	12	3	1.04	1.04
BitonicSort	452	676	4	1.253	1.178
DES	375	534	4	1.06	N/A

Evaluation - 4 cores

	4 cores		
Benchmark	k	APX/LR	APX/OPT
MergeSort	2	2.485	1.894
ChanVocoder7	2	1.729	1.506
MatrixMult	1	1.614	N/A
FFT2	1	1.826	N/A
FMRadio	2	1.639	N/A
BeamFormer	2	1.417	N/A
DCT	2	1.822	N/A
TDE	2	1.335	N/A
RadixSort	2	1.08	1.079
BitonicSort	2	1.397	N/A
DES	2	1.269	N/A

Evaluation - 6 cores

	6 cores		
Benchmark	k	APX/LR	APX/OPT
MergeSort	1	2.971	N/A
ChanVocoder7	1	1.349	N/A
MatrixMult	1	2.421	N/A
FFT2	1	2.739	N/A
FMRadio	1	1.775	N/A
BeamFormer	2	1.99	N/A
DCT	1	1.940	N/A
TDE	1	1.855	N/A
RadixSort	1	1.619	1
BitonicSort	2	1.851	N/A
DES	2	1.364	N/A

Evaluation - Linear Relaxation

Benchmark	LR	OPT
MergeSort	2.277	2.529
ChanVocoder7	155.185	157.495
MatrixMult	81.64	87.289
FFT2	33.872	34.284
FMRadio	8.272	N/A
BeamFormer	13.499	13.723
DCT	8.294	8.633
TDE	3183.437	3183.749
RadixSort	7.952	7.952
BitonicSort	3.056	3.25
DES	137.461	N/A

Summary

- Propose an approximation algorithm for APP that considers communicaton costs of data channels
- Proofs for NP-hardness, approximation bounds, and runtime
- Present instance bounds that provide better bounds than $\log n$ for a specific problem instance
- Evaluate our algorithm using StreamIt benchmark suite

References I

S. M. Farhad, Y. Ko, B. Burgstaller, and B. Scholz.
Orchestration by approximation: mapping stream programs onto multicore architectures.
In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS '11, pages 357–368, New York, NY, USA, 2011. ACM.

References II

S. M. Farhad, Y. Ko, B. Burgstaller, and B. Scholz.
Profile-guided deployment of stream programs on multicores.
In *Proceedings of the 13th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, Tools and Theory for Embedded Systems*, LCTES '12, pages 79–88, New York, NY, USA, 2012. ACM.

References III

M. Kudlur and S. Mahlke.
Orchestrating the execution of stream programs on multicore platforms.
In *PLDI '08: Proceedings of the 2008 ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2008.

References IV

H. Wei, J. Yu, H. Yu, and G. R. Gao.
Minimizing communication in rate-optimal software pipelining for stream programs.
In *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*, CGO '10, pages 210–217, New York, NY, USA, 2010. ACM.