Optimized On-Chip Pipelining of Memory-Intensive Computations on the CELL BE

Christoph Kessler
IDA / PELAB
Linköpings universitet
Linköping, Sweden

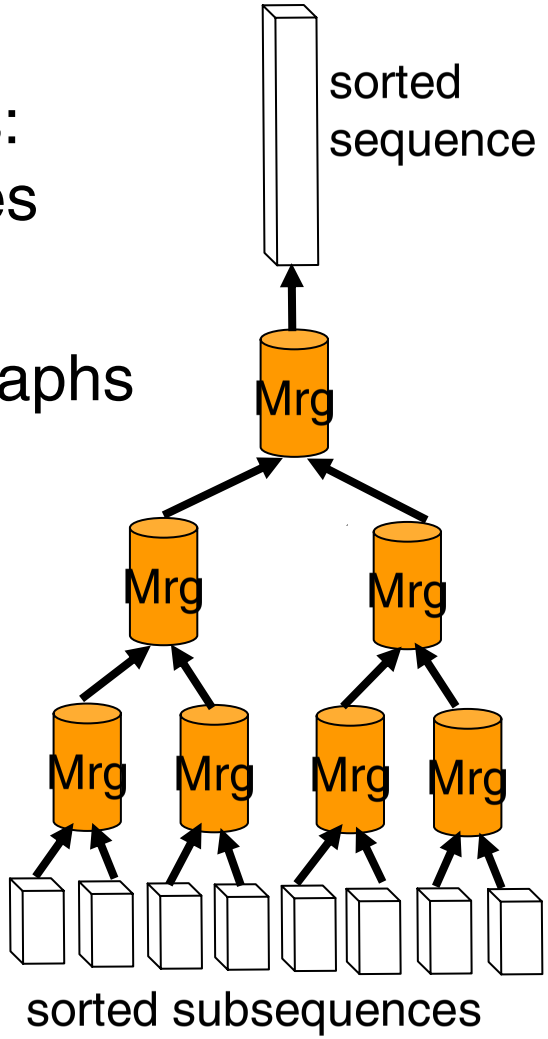
Jörg Keller
Dept. of Math. and CS
FernUniversität in Hagen
Hagen, Germany

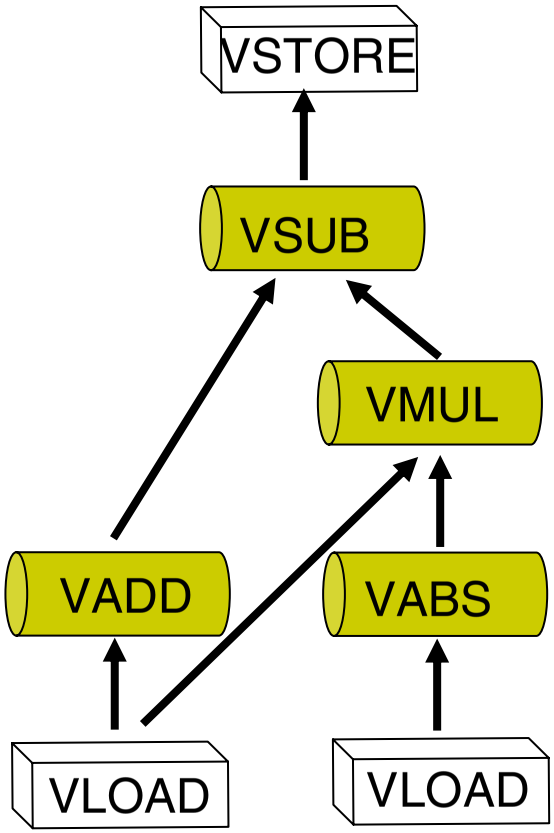
Outline

- Streaming Computations
- Cell BE Overview
- On-Chip Pipelining
- Optimization Problem
- ILP Solution
- Heuristic Solution
- Approximation Algorithm for Merge Trees
- Summary and Future Work

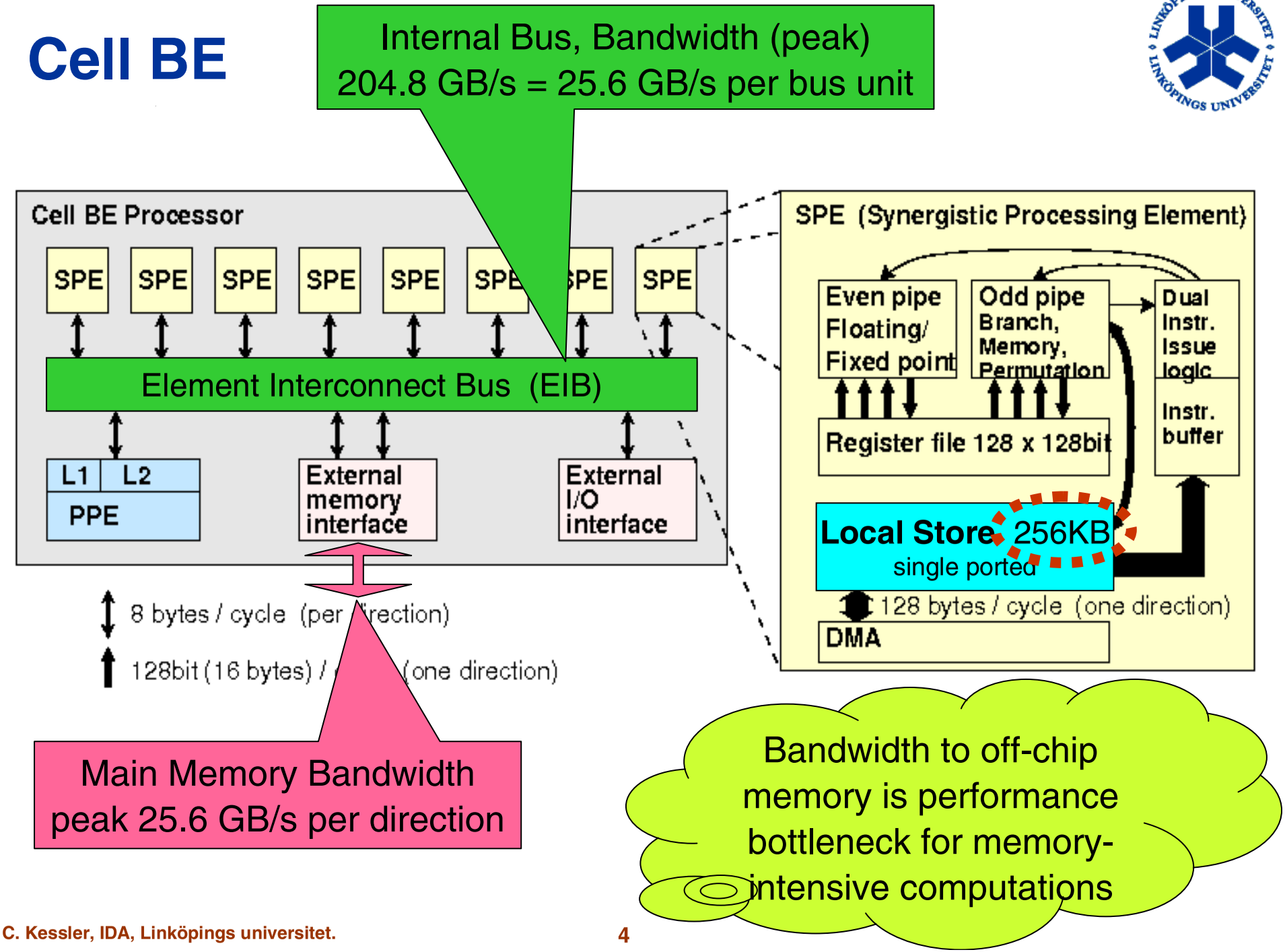
Streaming Computations

- Memory access intensive
- Given by task graph
  - Special topologies: linear chains, trees
  - General, directed acyclic graphs
- 2 kinds of tasks
  - serial, e.g. merge
  - data-parallel (vector ops)





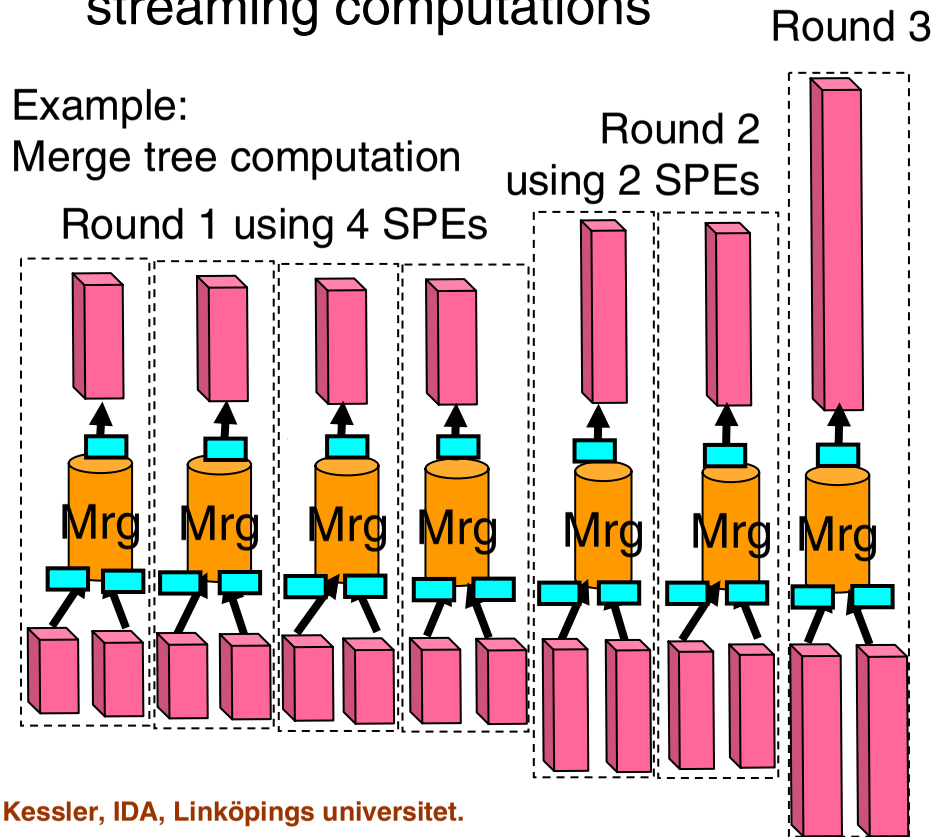
Cell BE


Internal Bus, Bandwidth (peak)
204.8 GB/s = 25.6 GB/s per bus unit
Cell BE Processor
SPE
SPE
SPE
SPE
SPE
SPE
SPE
SPE
Element Interconnect Bus (EIB)
L1
L2
PPE
External memory interface
External I/O interface
8 bytes / cycle (per direction)
128bit (16 bytes) / (one direction)
Main Memory Bandwidth
peak 25.6 GB/s per direction
SPE (Synergistic Processing Element)
Even pipe
Floating/
Fixed point
Odd pipe
Branch,
Memory,
Permutation
Dual
Instr.
Issue
logic
Instr.
buffer
Register file 128 x 128bit
Local Store 256KB
single ported
128 bytes / cycle (one direction)
DMA
Bandwidth to off-chip memory is performance bottleneck for memory-intensive computations

Orchestrating SPEs and main memory

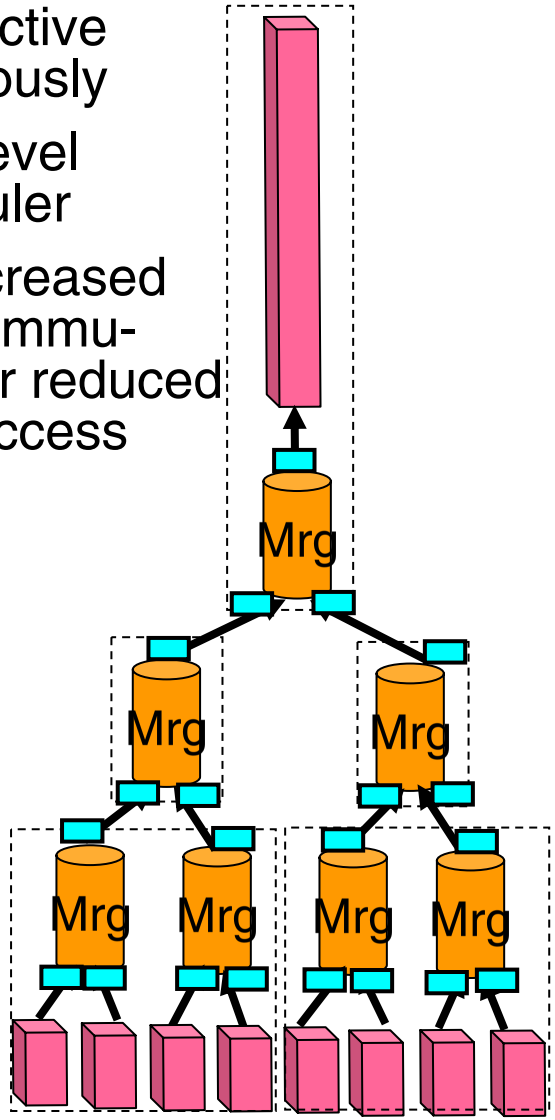
Dancehall

- Use SPE local store as explicitly managed cache for shared main memory
- Bandwidth to main memory is performance bottleneck for streaming computations



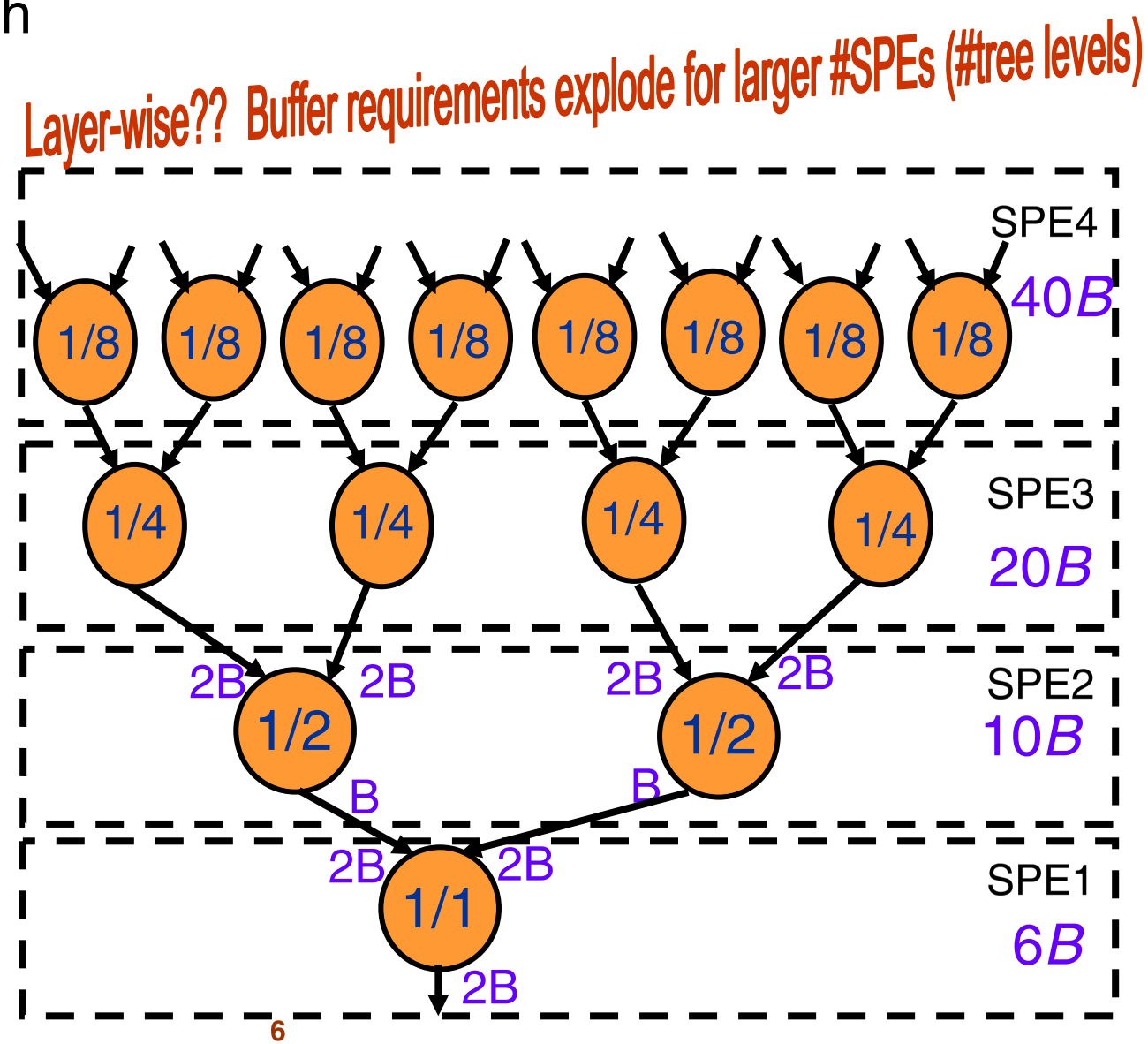
On-Chip Pipelining

- All tasks active simultaneously
  - User-level scheduler
- Trades increased on-chip commu-nication for reduced memory access



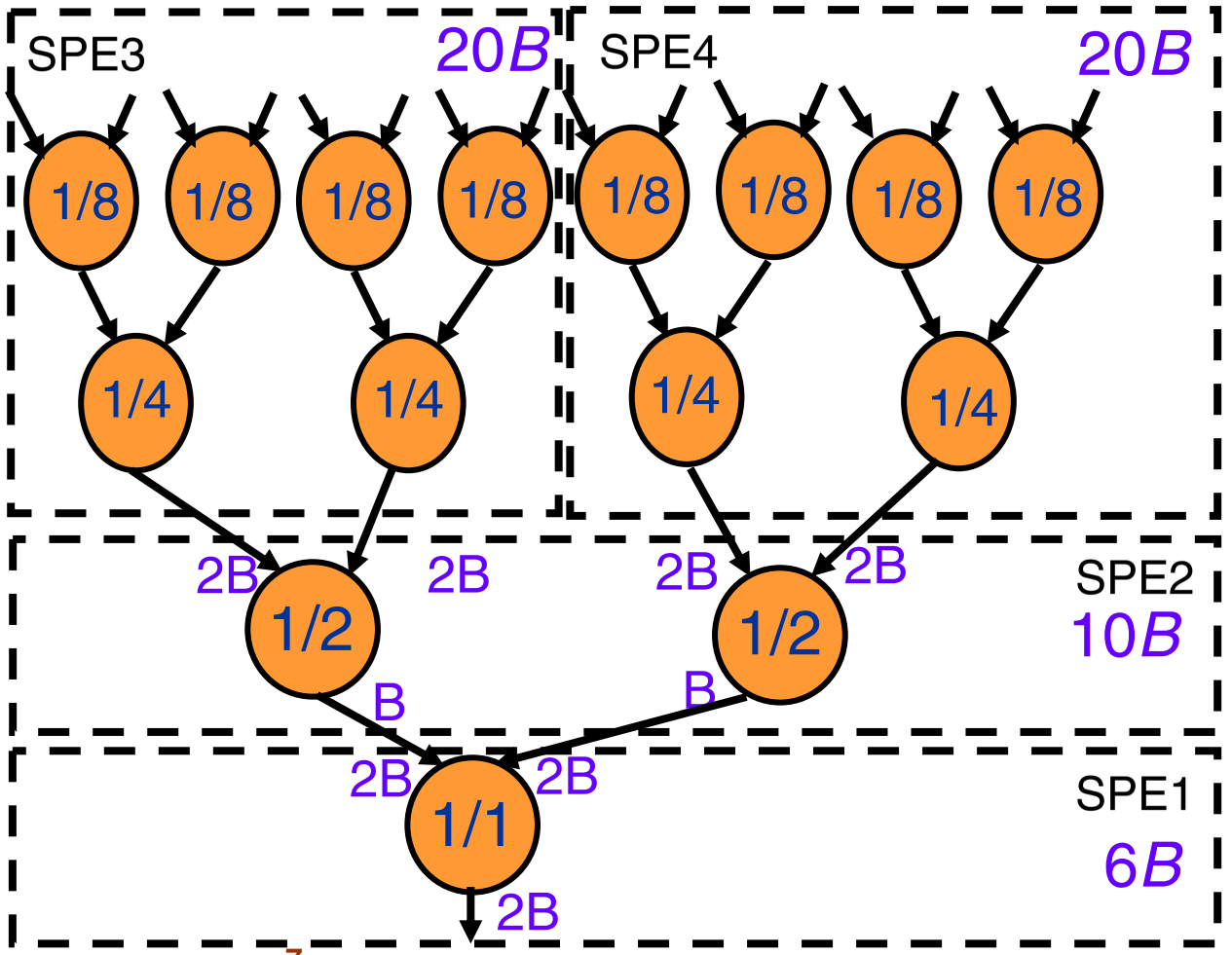
Case Study: On-Chip-Pipelining for Merger Trees (1)

- Pipeline taskgraph
- Double-buffering
- Example: Map 16-to-1 Merger-tree (4-level tree) to 4 SPEs
- Tasks:
  - Comput. load
  - Memory load
  - Comm. rate
- Which mappings are good?

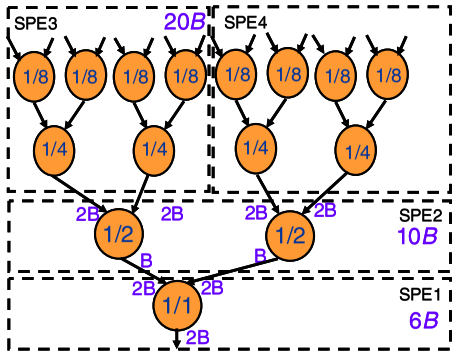


Case Study: On-Chip-Pipelining for Merger Trees (2)

- Pipeline taskgraph
- Double-buffering
- Example: Map 16-to-1 Merger-tree (4-level tree) to 4 SPEs
- Tasks:
  - Comput. load
  - Memory load
  - Comm. rate
- Which mappings are good?



Optimization Problem

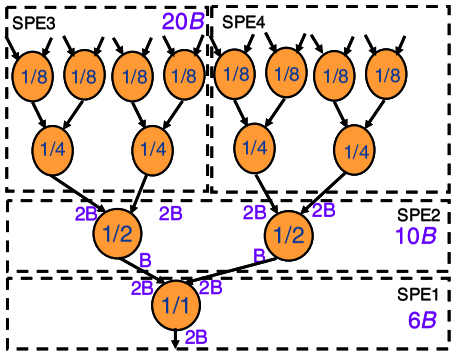


- Max. accumulated work load mapped to a SPE should be minimal
- Max. Σ buffers per SPE should be minimal
  - as DMA performance degrades for small packet sizes and local store is very small
- Inter-SPE communication volume should be low (max. 204 GB/s)

→ Multi-objective optimization problem

ILP Model

- $x_{v,q} = 1$
  iff task *v* mapped to SPE *q*



$$\forall v \in V: \quad \sum_{q \in P} x_{v,q} = 1$$

$$\forall q \in P: \quad \sum_{v \in V} x_{v,q} \cdot \rho(v) \leq maxComputLoad$$

$$\forall q \in P: \quad \sum_{v \in V} x_{v,q} \cdot \beta(v) \leq maxMemoryLoad$$

$$\forall (u,v) \in E,\ q \in P: \quad z_{(u,v),q} \leq x_{v,q}$$

$$z_{(u,v),q} \leq x_{u,q}$$

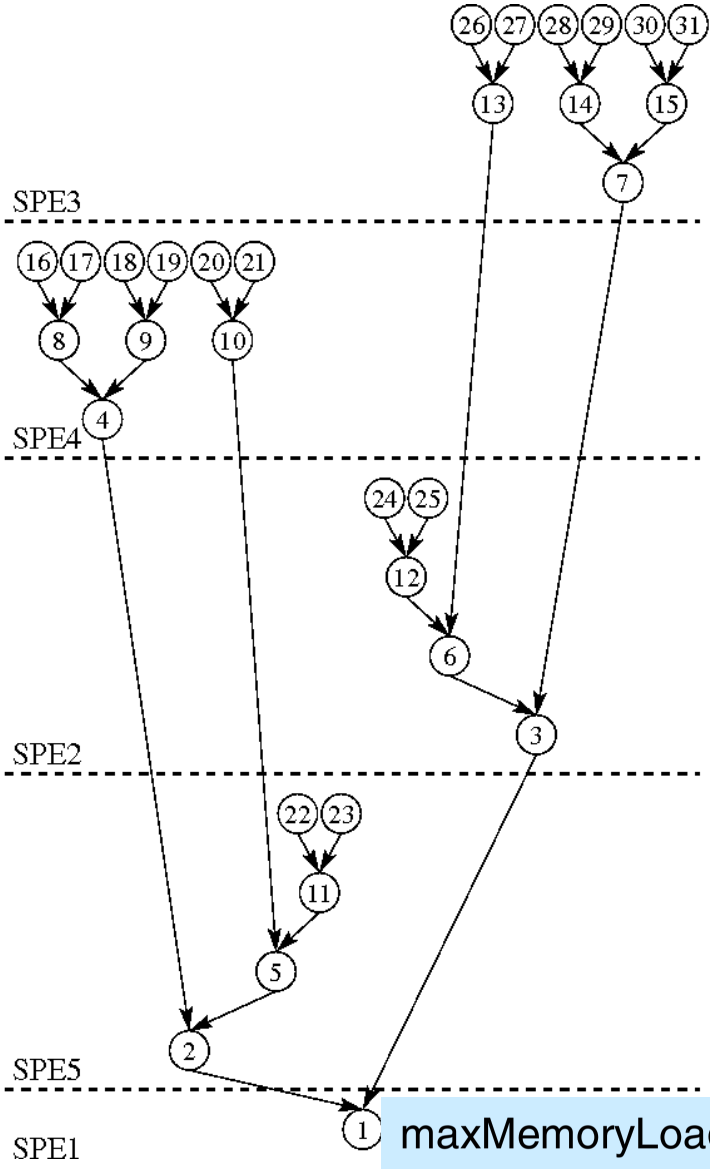
$$commLoad = \sum_{e \in E} \tau(e) - \sum_{e \in E} \sum_{q \in P} z_{e,q} \cdot \tau(e)$$

Minimize $\Lambda \cdot maxComputLoad + \epsilon_M \cdot maxMemoryLoad + \epsilon_C \cdot commLoad$

where $0 \leq \epsilon_M < 1$ and $0 < \epsilon_C < 1$

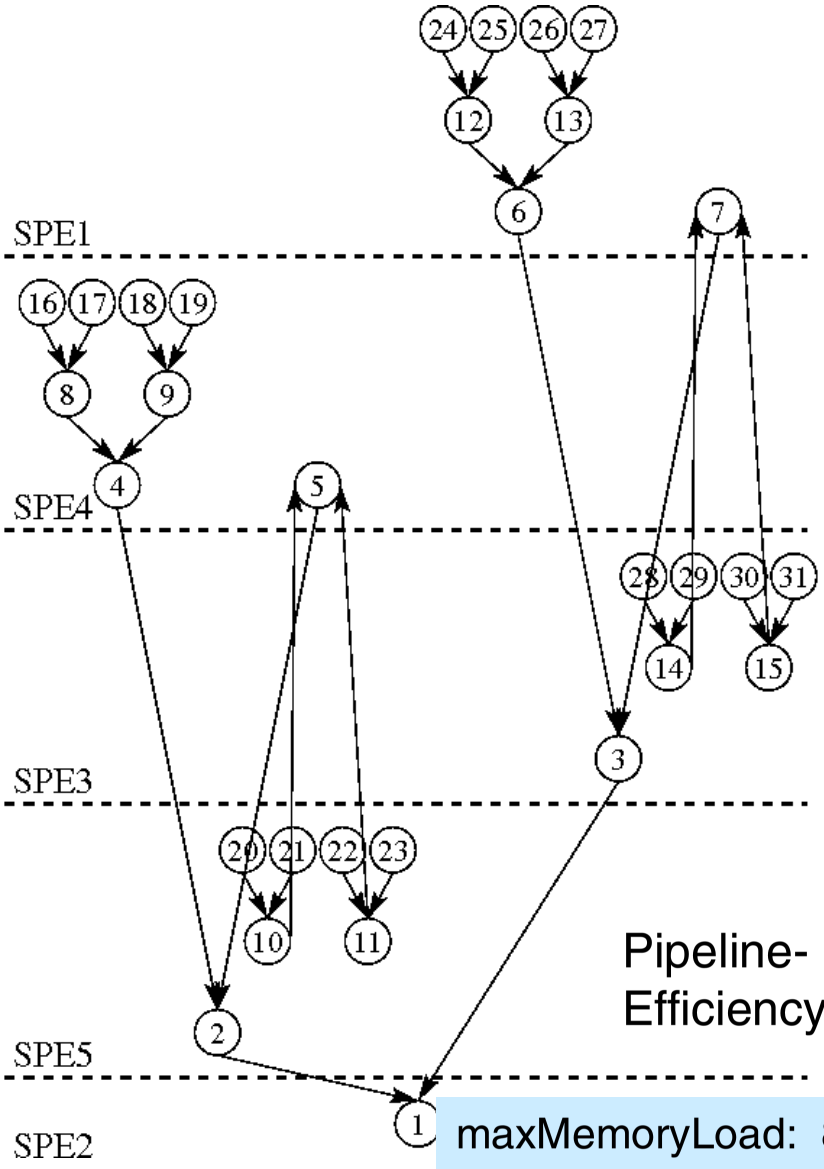
O(*np*) variables
O(*np*) constraints

ILP Mapping Results for Merge Trees:
2 Pareto-optimal Mappings (32-to-1 Merger)



maxMemoryLoad: 10x5*B*

maxInterSPEComm: 1.75



Pipeline-Efficiency 93%

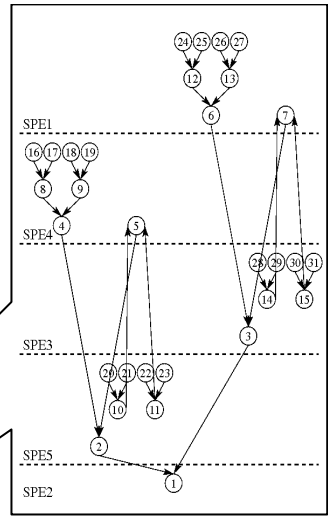
maxMemoryLoad: 8x5*B*

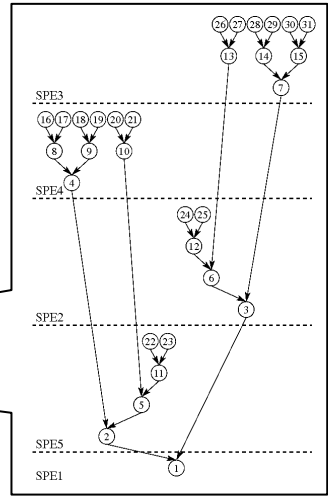
maxInterSPEComm: 2.5

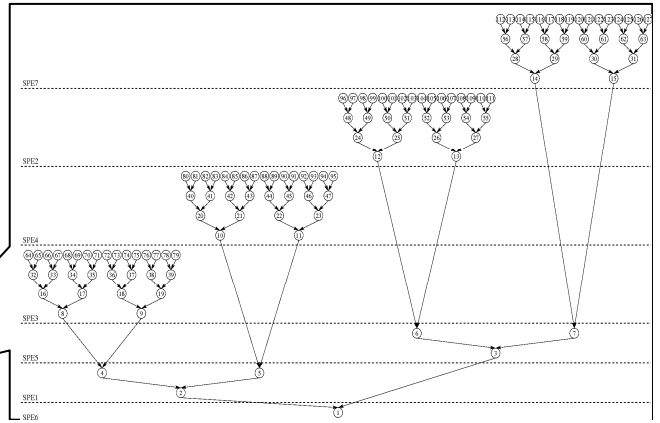
More ILP Results for Merger Trees

Table 1. The Pareto-optimal solutions for mapping b-ary merge trees, found with ILP, for $b = 2$, $k = p = 5,\ 6,\ 7$.

k	binary variables	constraints	*max. memory load*	*commLoad*
$k = 5$	305	341	8	2.5
			9	2.375
			10	1.75
$k = 6$	750	826	13	2.625
			14	2.4375
			15	1.9375
			20	1.875
$k = 7$	1771	1906	21	2.375
			29	2.3125
			30	2



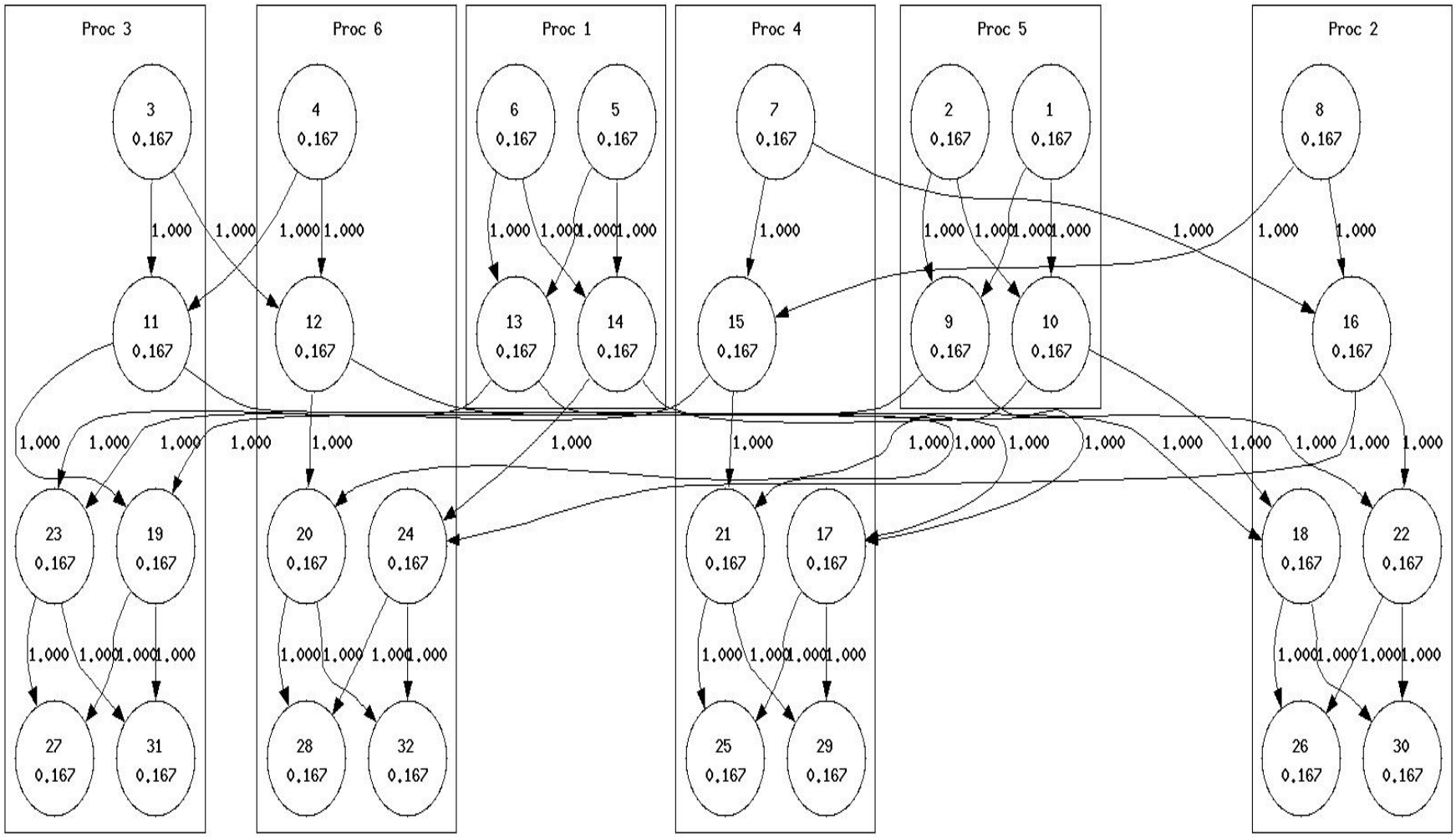




C.

ILP Mapping Result for FFT

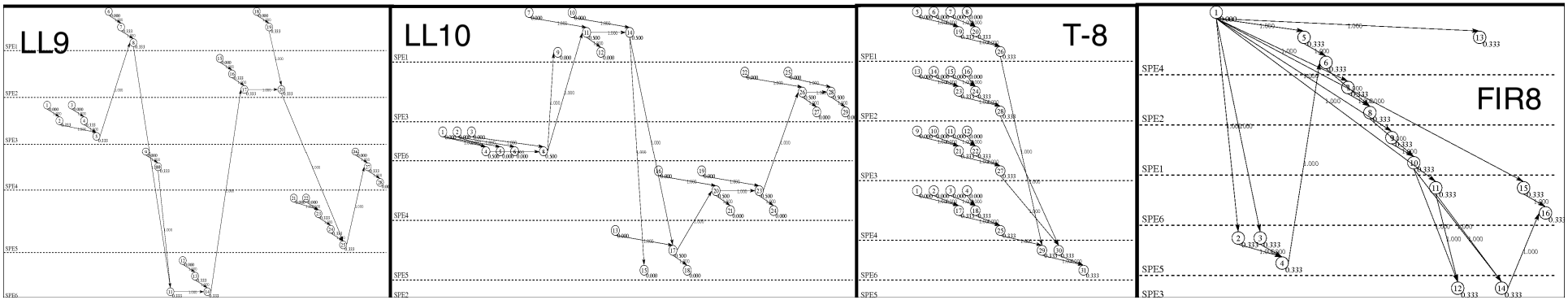
- 8x4 FFT (Butterfly task graph) on 6 SPEs



ILP Mapping results for general data-parallel task graphs

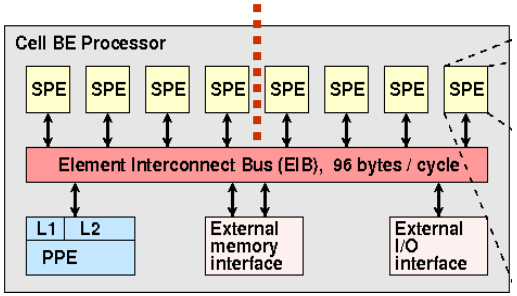
- Data-parallel task graphs from vectorized Livermore Loops and synthetic benchmarks

Kernel	Description	#Nodes	#Edges	ILP model for $p = 6$			ILP model for $p = 8$		
		n	m	var's	constr.	time	var's	constr.	time
LL9	Integrate predictors	28	27	333	371	2:07s	443	485	—
LL10	Difference predictors	29	28	345	384	0:06s	459	502	1:26:39s
LL14	1D particle in cell, 2nd loop	19	21	243	290	0:03s	323	380	1:05s
LL22	Planckian distribution	10	8	111	125	<0:01s	147	163	<0:01s
FIR8	8-tap FIR filter	16	22	231	299	45:04s	307	393	0:04s
T-8	Binary tree, 16 leaves	31	30	369	410	5:36s	491	536	0:11s
C-6	Cook pyramid, 6 leaves	21	30	309	400	27:56s	411	526	3:22s

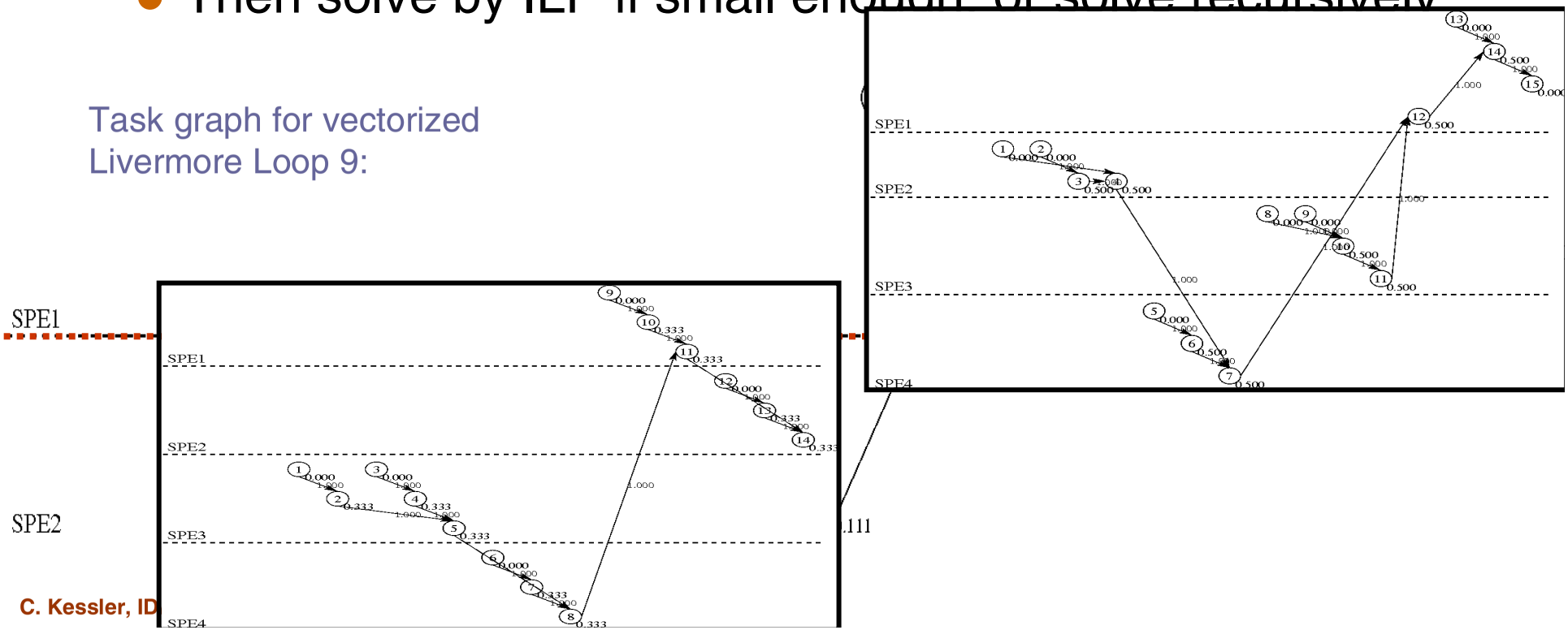


Divide-and-Conquer Heuristic

- ILP complexity grows linearly in *p* and *n*
  - Too high for p>=8 on larger task graphs
- Split both task graph and SPE set in halves, using ILP model for p=2
  - Then solve by ILP if small enough, or solve recursively

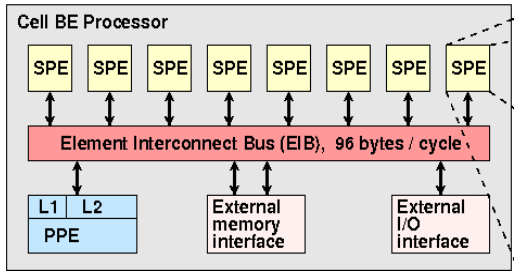


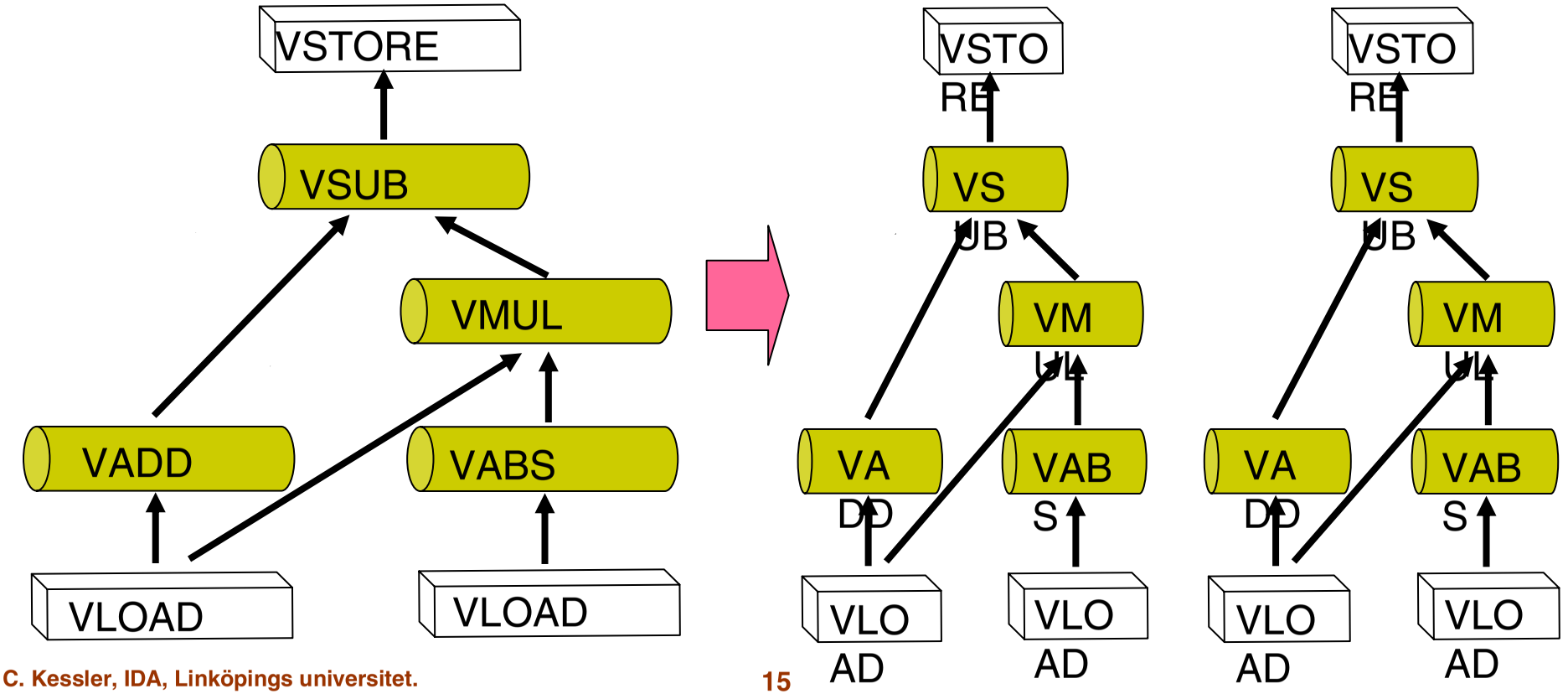
Task graph for vectorized Livermore Loop 9:



Granularity Control

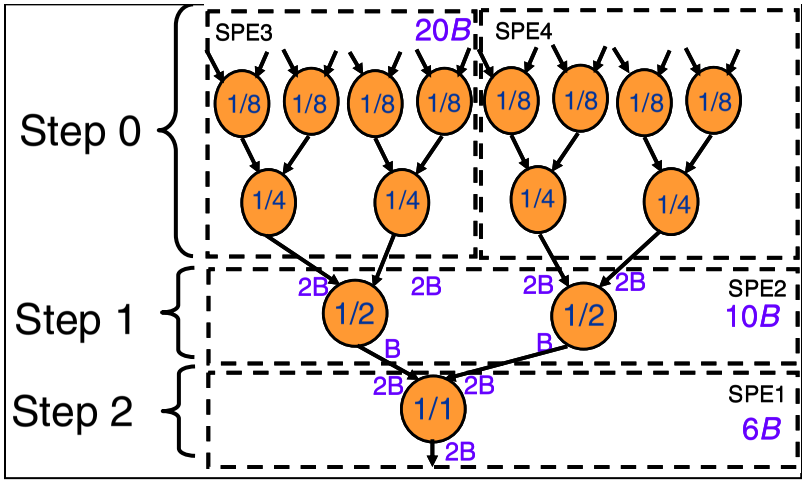
- Too few heavy-weight tasks → not all SPEs utilized
- Solution: Strip-Mining (Task splitting)
  - Only for data-parallel tasks

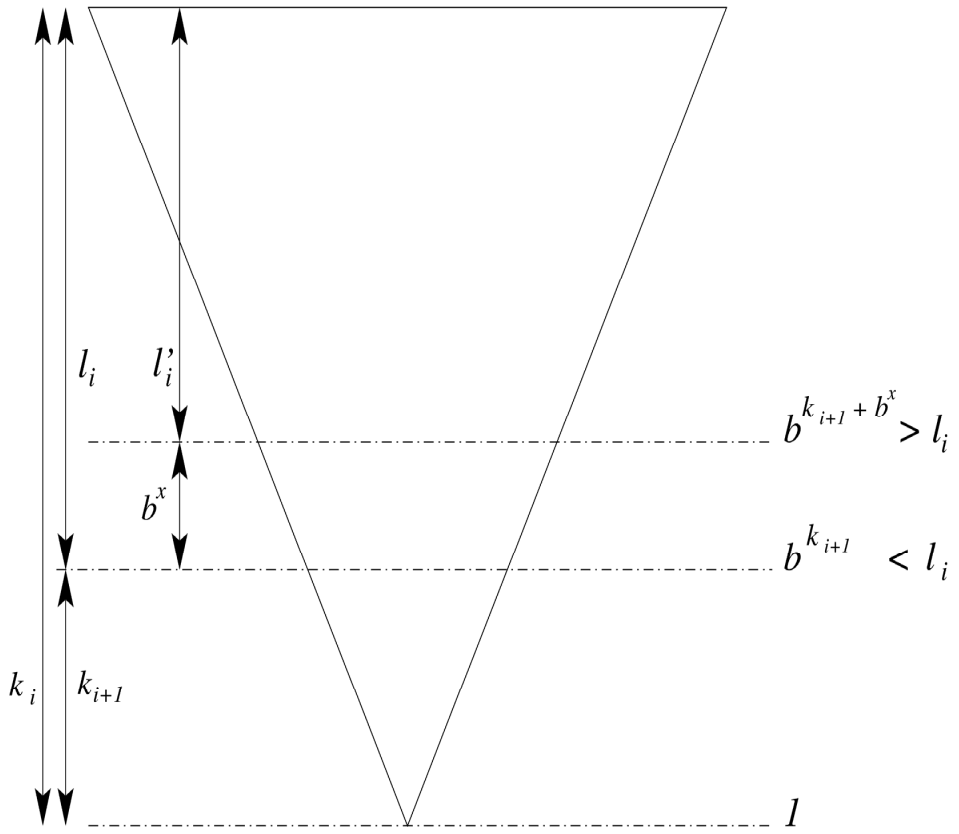




Approximation Algorithm for *b*-ary Merge Trees

- Iterative algorithm
  - Maps a *k*-level *b*-ary merge tree to $p=k$ SPEs in $O(b \log k)$ steps
- Step *i* maps l_i lowest yet unmapped levels to l_i SPEs
  - where l_i is the largest power of *b* that is <= the number of yet unmapped levels – 1
  - Mapping by equal distribution of subtrees
- Details: in the paper



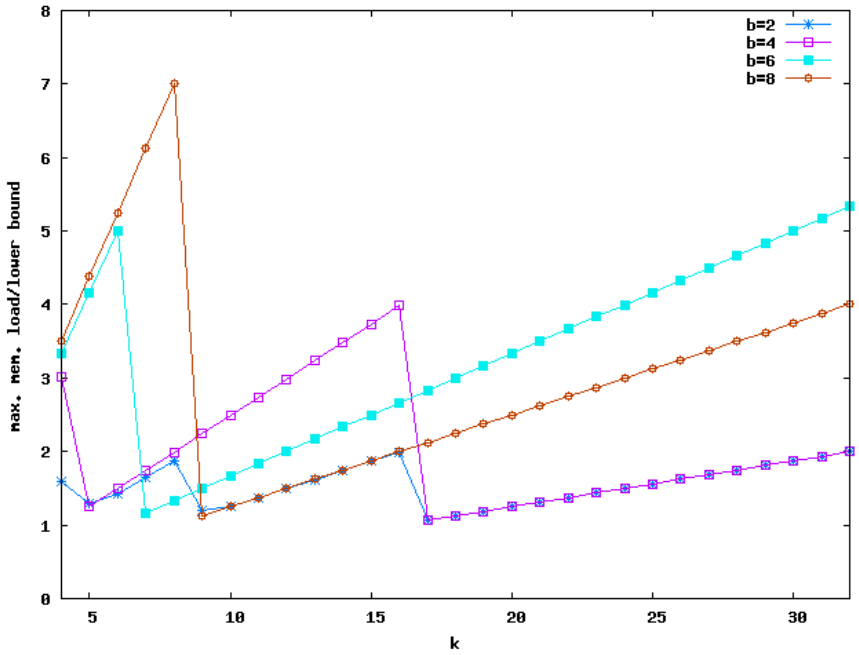


Approximation Algorithm for *b*-ary Merge Trees (2)

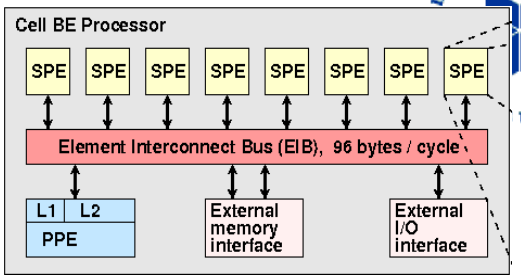
The computed mapping has the following properties:

- The maximum computational load is 1/1 (= root merger load)
  - This is optimal
- The maximum memory load is about $\frac{b^k - b^{k-l_0}}{(b-1)l_0} B$
  - This is larger than a straightforward lower bound by a factor < *b*

Proof: in the paper



Summary



Optimized On-Chip Pipelining for CELL

- Trade more internal communication for less memory accesses
- Optimize mapping of tasks for computational load balance, SPE memory load balance, overall communication volume
- ILP model for general task graphs
- DC-heuristic for general task graphs
- Iterative approximation algorithm for merge trees

Future Work

- Implementation to verify simulation results