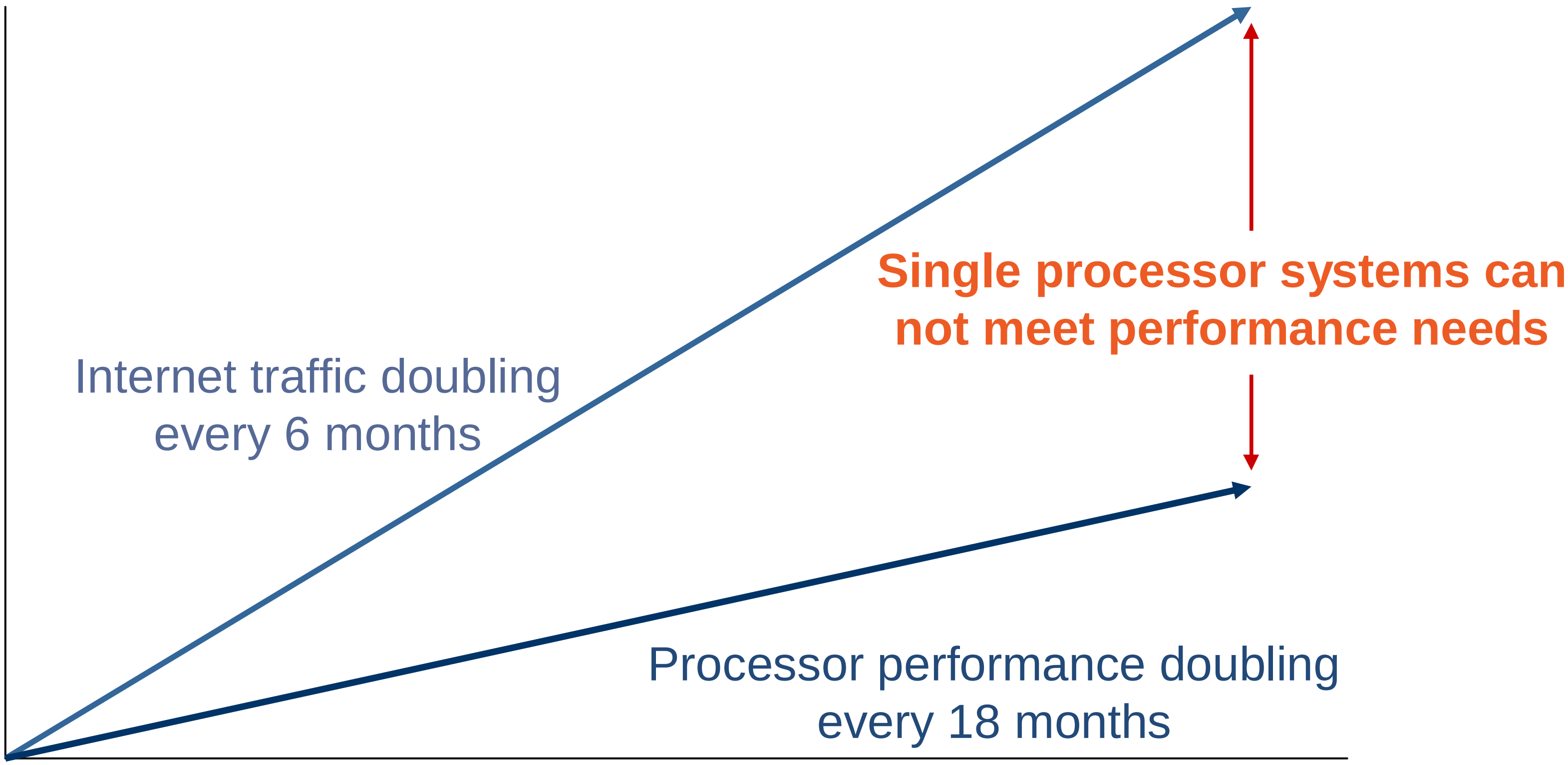
Approximation Algorithm for Process Mapping on Network Processor Architectures

Chris Ostler, Karam S. Chatha,
Goran Konjevod

Department of Computer Science
Arizona State University

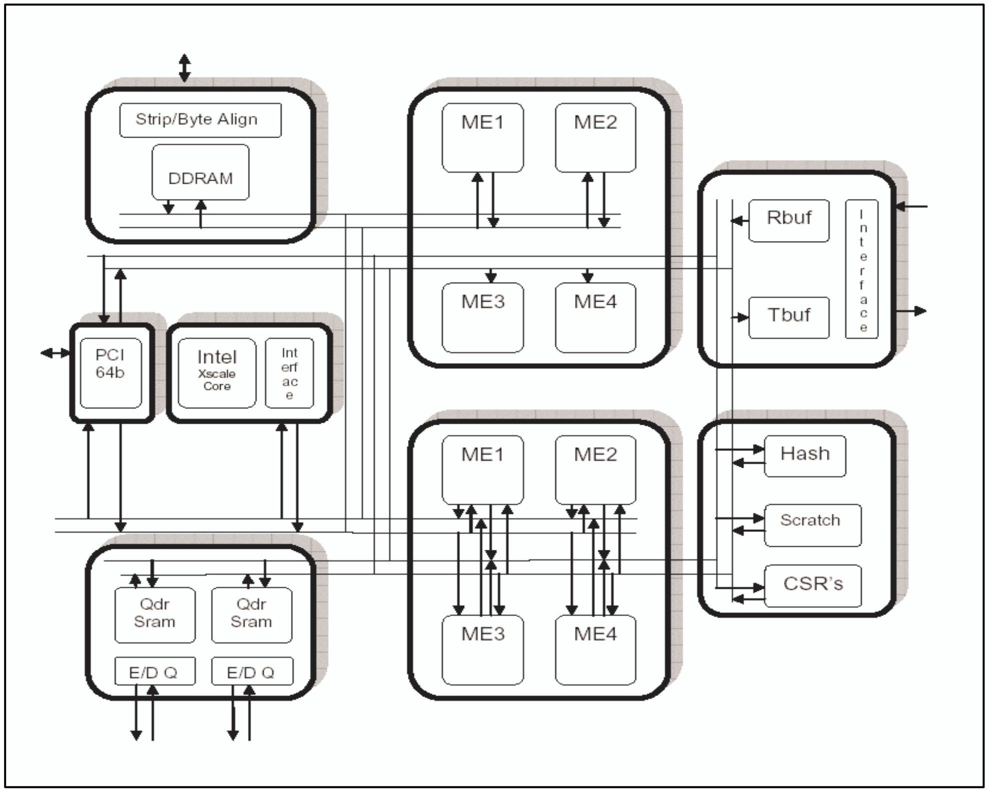
Traffic Processing Growth



Lead to the development of *multi-core, multi-threaded* network processor architectures

Intel IXP 2400 Processor

- Eight independent micro-engines
  - Support for 8 threads
  - Block execution
- Available memory
  - 2.5 KB local memory
  - 16 KB scratchpad
  - Off-chip SRAM
  - Off-chip DRAM



- Complex architecture that is challenging to program in absence of structured methodology

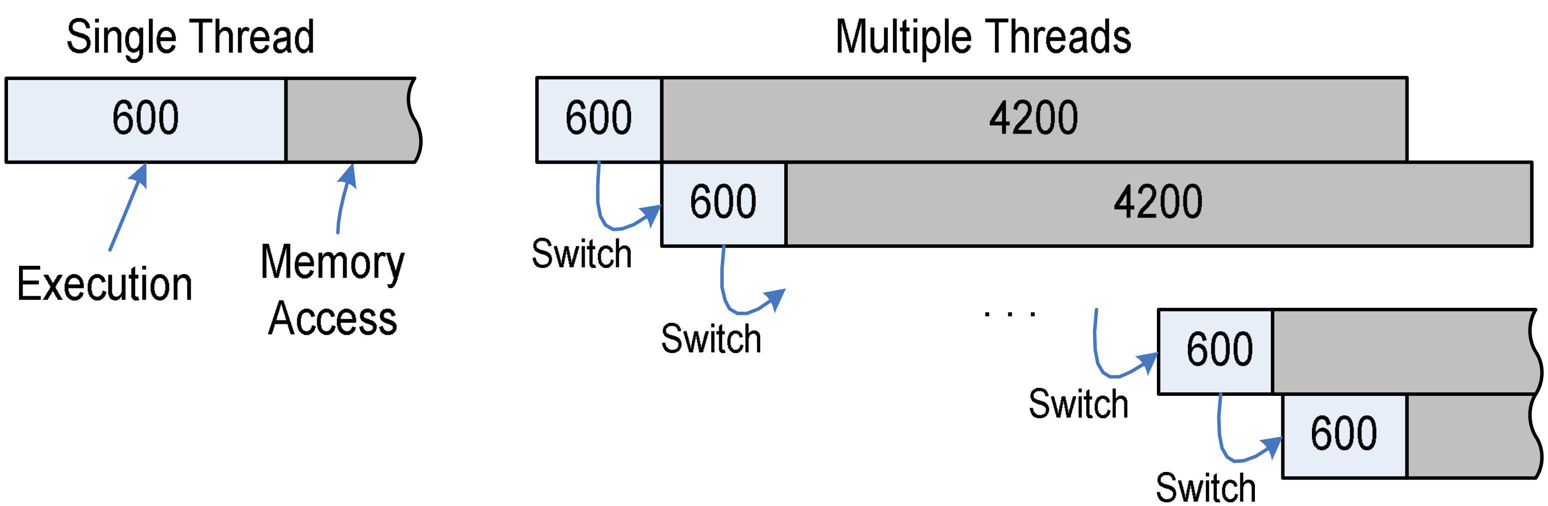
Paper Contributions

- Develop throughput optimization strategies
  - Process network transformations
- Discuss properties of optimal solutions
  - Derive upper bound on throughput
- Propose approximation algorithm
  - Results have throughput at least ½ optimal

Multi-threading and Memory Latency

For 600 cycles of execution, 4200 cycles of latency:

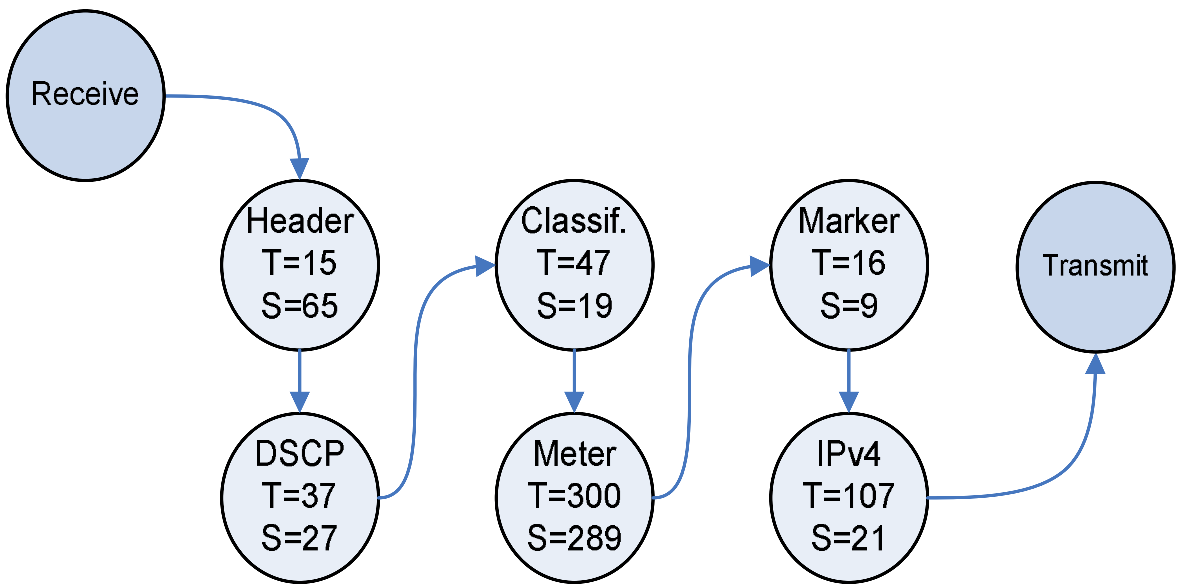
- Single thread completes once every 4800 cycles
- Multiple (8) threads complete once every 600 cycles



Memory latency can be hidden by multi-threading

- Possible to ignore memory latency
- Consider only effect of execution time on throughput

Application Description



Data Items
Packet store
DSCP table
. . .
Meter table
Routing table

Process network specification

- Concurrently executing processes
- Communicate only through bounded FIFOs
- May use abstract shared memory (tables, etc.)
- Profiled to determine code size, execution time

Problem Description

Given:

- Set J of jobs, each characterized with execution time t_j, code size s_j
- M symmetric processors, with MEM_AVAIL instruction memory

Objective:

- Find static mapping of jobs to machines to maximize worst case throughput

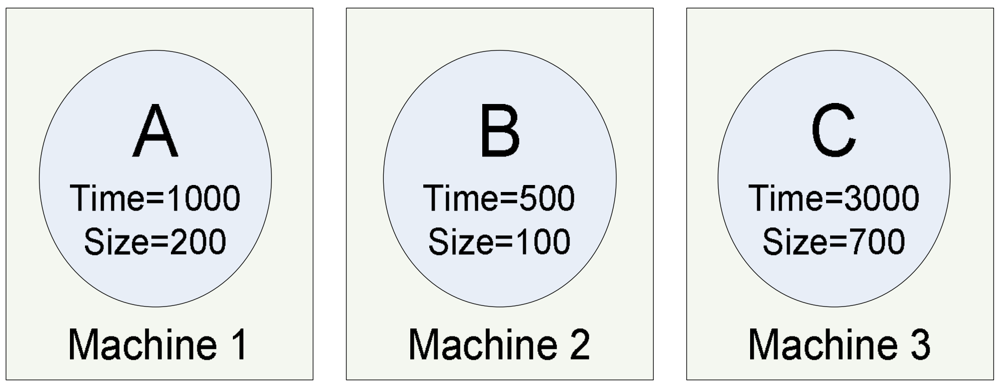
Such that:

- All jobs assigned to a machine fit in available memory

Motivating Example

Consider simple case:

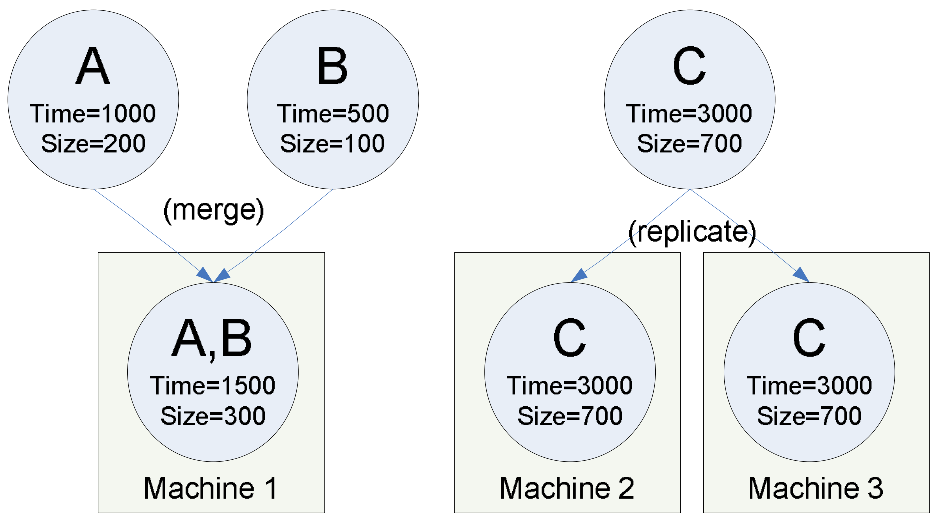
- 3 jobs, 3 machines
- Assign one job to one machine



Throughput is one completion every 3000 cycles

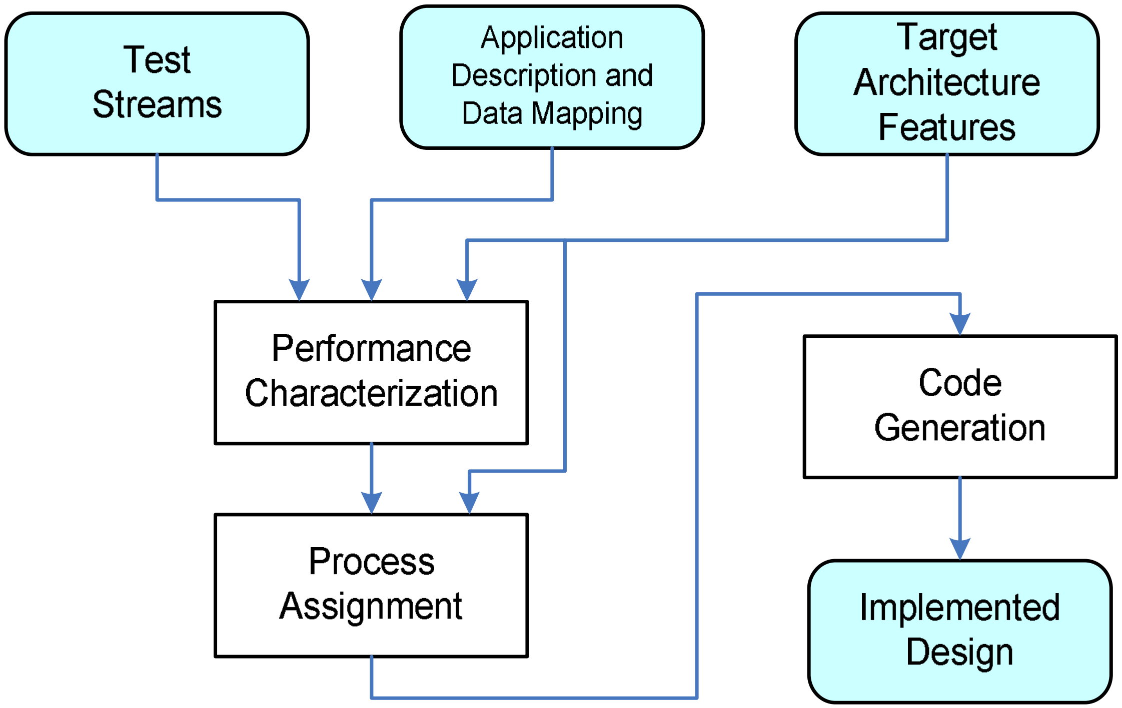
Merge and Replicate Transformations

- Alter process network to increase throughput
- Eliminate highest, lowest throughput processes



Throughput is doubled, but transformations are limited by code memory

Design Flow



Focus of paper is process assignment

Require as input:

- Application description
- Data mapping
- Performance characterization
- Architectural features

Previous Work

- Task Allocation
  - Shirazi et al. (1995)
  - *Not applicable to network processor architectures*
- Scheduling Synchronous Dataflow Networks
  - A. Jantsh (2005)
  - Sriram et al. (2000)
  - *Do not consider code memory constraints*
  - *Minimize latency not optimize throughput*
- Network Processor Techniques
  - Shah et al. (2002)
  - Ramaswamy et al. (2005)
  - *Do not exploit parallelism of applications*

Optimal Solutions

- Best throughput when execution divided evenly among all machines

$$\text{Throughput} = \frac{M}{\sum_{j \in J} t_j}$$

- Every machine has 100% utilization
- Can be achieved by assigning all jobs to every machine

Job Assignment Algorithm

- Given a set of jobs, we can determine the ideal fraction (x_j) of available machines

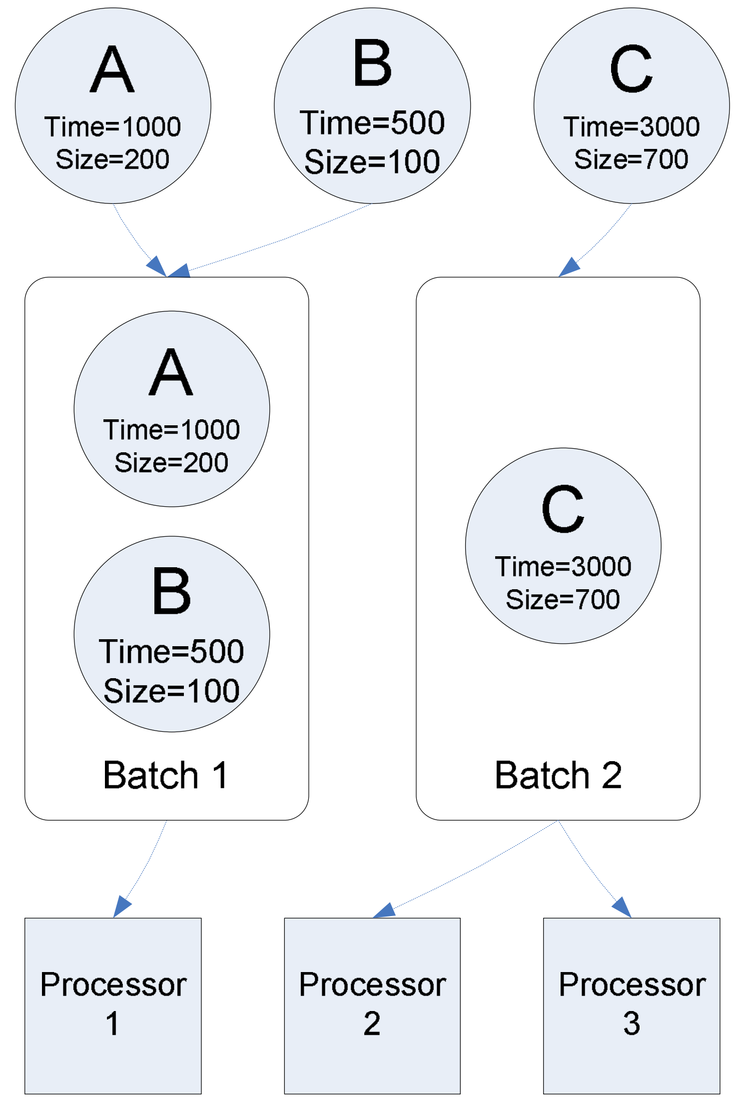
$$x_j = \frac{M \cdot t_j}{\sum_{i \in J} t_i}$$

- Simple strategy: round down x_j, except *cannot assign a job zero machines*
  - Too many jobs ($|J| > M$) $\rightarrow$ Batching
  - Some jobs $x_j \leq 1$ $\rightarrow$ Recursive strategy

Batching

- Assign jobs to batches
- Assign batches to machines

- Incorporates merge, replicate transformations
- Reduces problem complexity
- Optimal throughput remains unchanged



Recursive Solution Strategy

- Batch jobs, and determine x_b for all batches
- Find batch s with smallest x_s
  - If $x_s \geq 1$, round down all x_b
  - Otherwise, round up x_s, solve for remaining jobs, machines

Jobs must be batched so that final throughput can be guaranteed

MAX_MIN_TIME Function

Given:

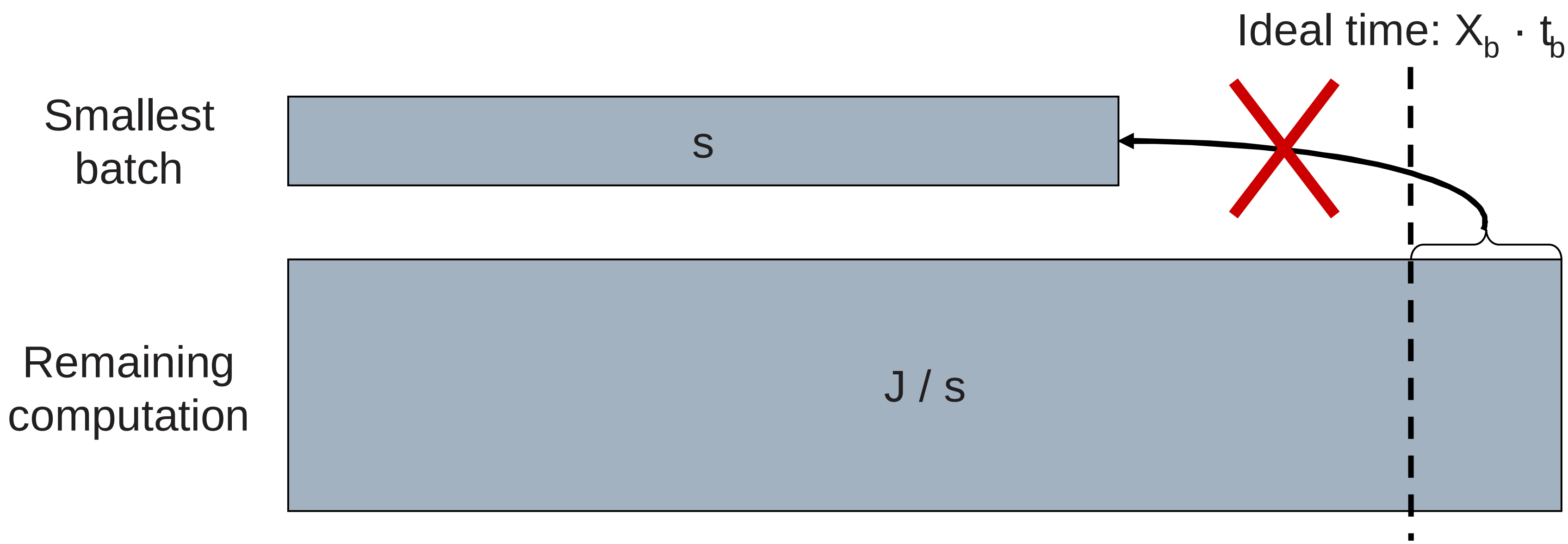
- A set of jobs, with execution time and size
- A number of batches
- The amount of code memory available

Objective:

- Assign jobs to batches so that the minimum execution time is maximized

Currently implemented using simple ILP

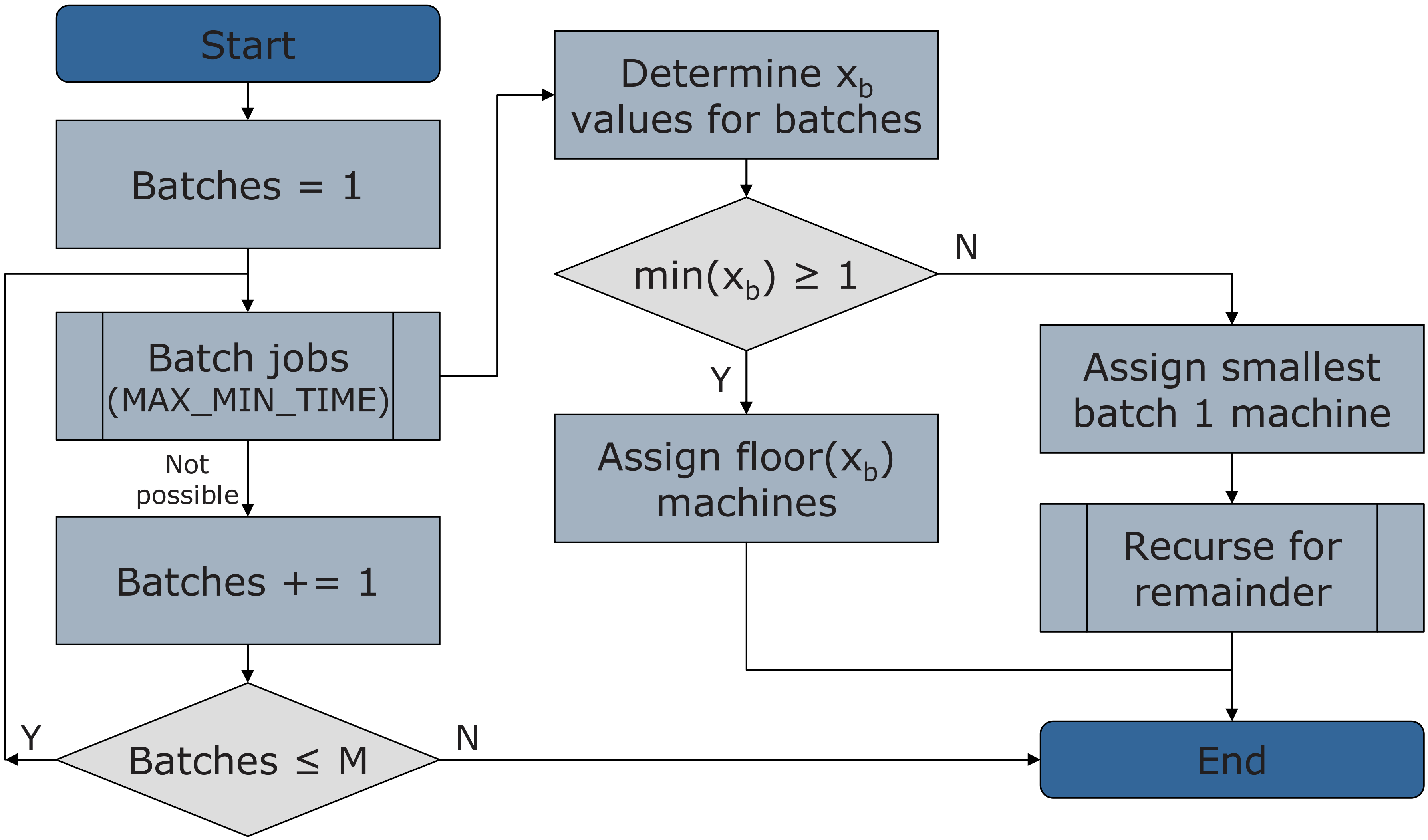
MAX_MIN_TIME Function



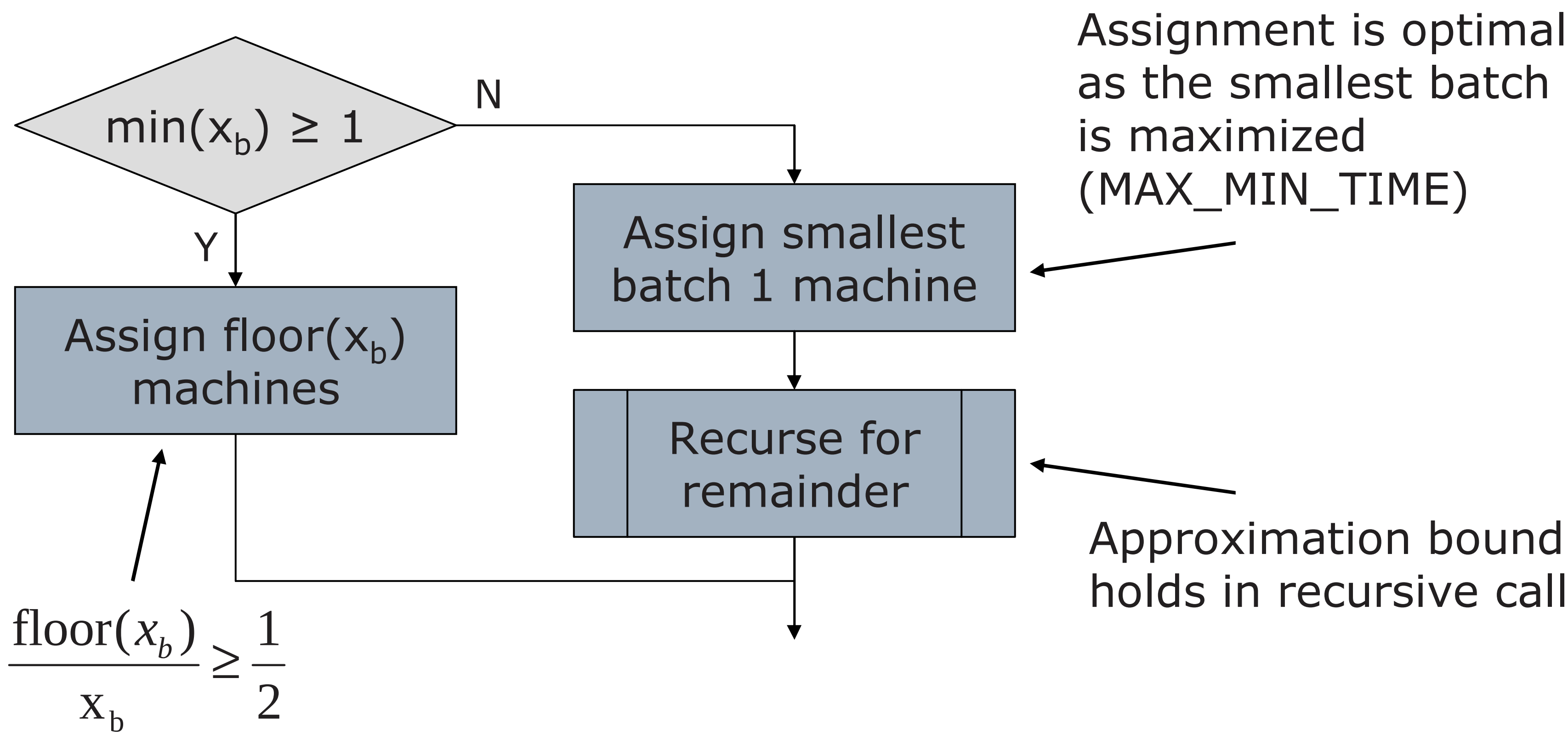
No more computation can assigned small batch

- Machine will be under-utilized vs. ideal
- Same under-utilization in an optimal solution
- Machine assignment is optimal

ASSIGN_JOBS Algorithm



Approximation Bound



Each batch assigned at least ½ ideal machines

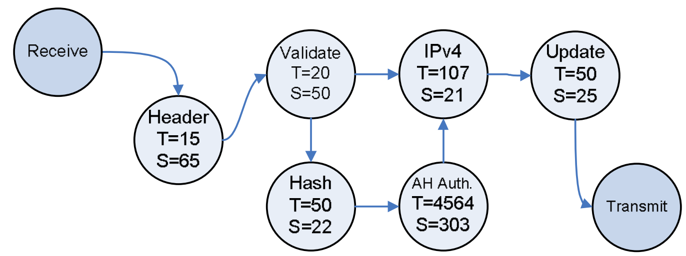
- Complete in no more than twice optimal time
- *Overall throughput at least ½ optimal*

Experimental Results

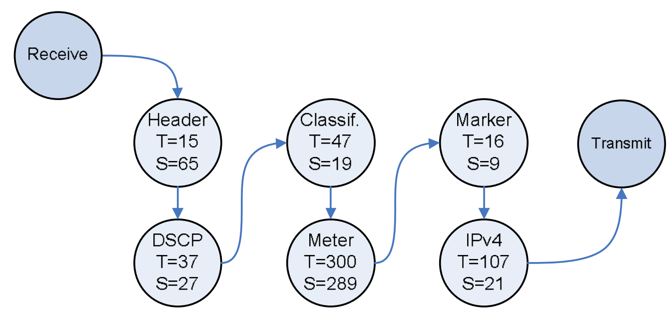
- Use algorithm to map three common network processing applications
- Targeted Intel IXP 2400 network processor
  - Limit code memory to 400 instructions
  - Reserved 2 micro-engines for receive, transmit
- Compare solution throughput to upper bound, ILP formulation using batching
  - Runtime of non-batched ILP prohibitive

Experimental Applications

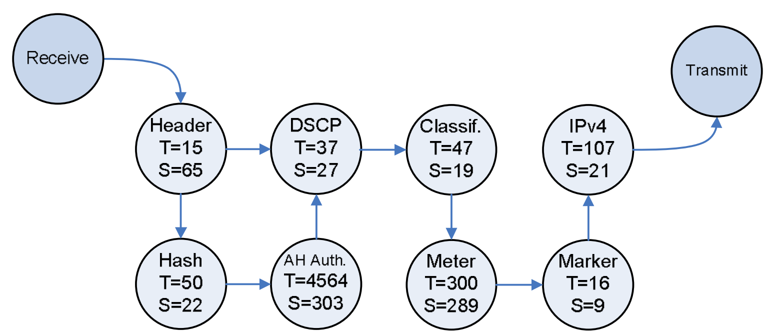
- IPSec, IPv4
  - 6 processes
  - Diverse execution time



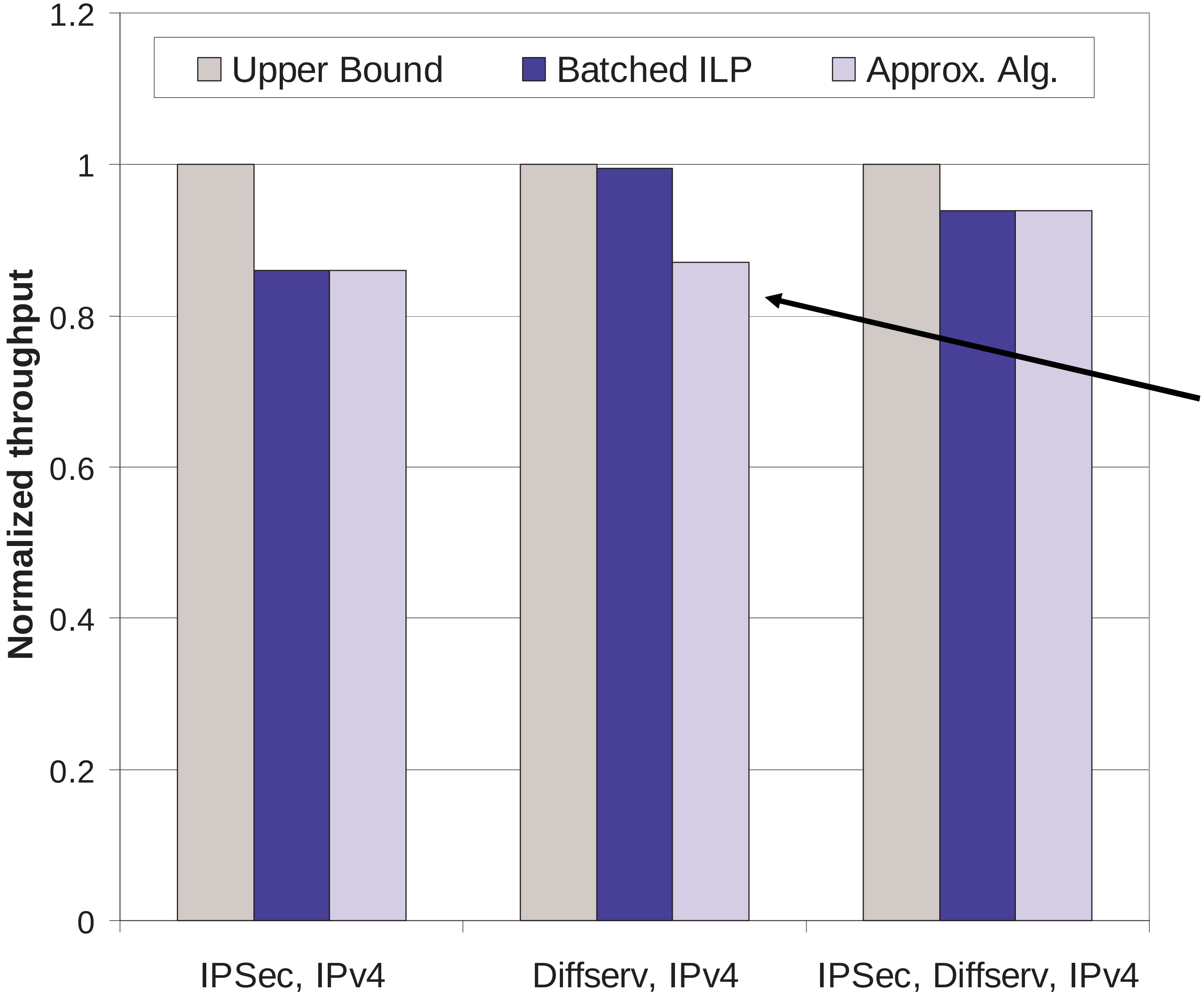
- Diffserv, IPv4
  - 6 processes
  - Uniform execution time



- IPSec, Diffserv, IPv4
  - 8 processes
  - Diverse execution time



Mapping Results



Throughput within:
- 78% of bound
- 87% of ILP

Throughput normalized to upper bound

Conclusion

- Tools are necessary to fully exploit multi-core network processors
- Proposed approximation algorithm to map application to processing cores
  - Solutions guaranteed to have throughput at least half that of optimal solution
  - Experimental results showed throughput within 78% of optimal