Mapping Vector Codes to Stream Processor (Imagine)

Mehdi Baradaran Tahoori

Paul Wang Lee

June 4th, 2002

Outline

- Motivation
- Problem Statement
- Simulation Results
- Conclusion

Motivation

- Large volume of vector code in existence
  - Arithmetic intensive
- Much existing research on vectorization
  - Vectorizing compilers, etc.
- Stream programming
  - Intermediate data
    - Producer-consumer locality
    - Shorter lifetime than in vector processor

Problem Statement

Efficient mapping of vector codes to stream processor

- **Pseudo vector code**
  - **Not focusing on syntax**
- **Focus on specific hardware**
  - **Imagine architecture**
  - **Imagine programming model**
    - **Stream C & kernel C**
      - **No performance evaluation in Brook**

Goals

- **Maximize resource utilization**
- **Minimize memory bandwidth requirements**
  - **SRF $\leftrightarrow$ LRF**
  - **SRF $\leftrightarrow$ μC**
- **Minimize inter-cluster communications**
  - **Specially for vector reduction operations**
    - **Inner-product, matrix $\times$ vector, …**

Approach

- Implementation in KernelC & StreamC
  - Cycle accurate simulation
    - Various representative code snippets
    - Various record sizes
    - Various kernel granularity
  - Considering realistic settings
- Observations through simulation
  - Interpret results
  - Look for rules
    - Can be applied to mapping strategy

Partitioning

- Modulo Data
  - Stream element size

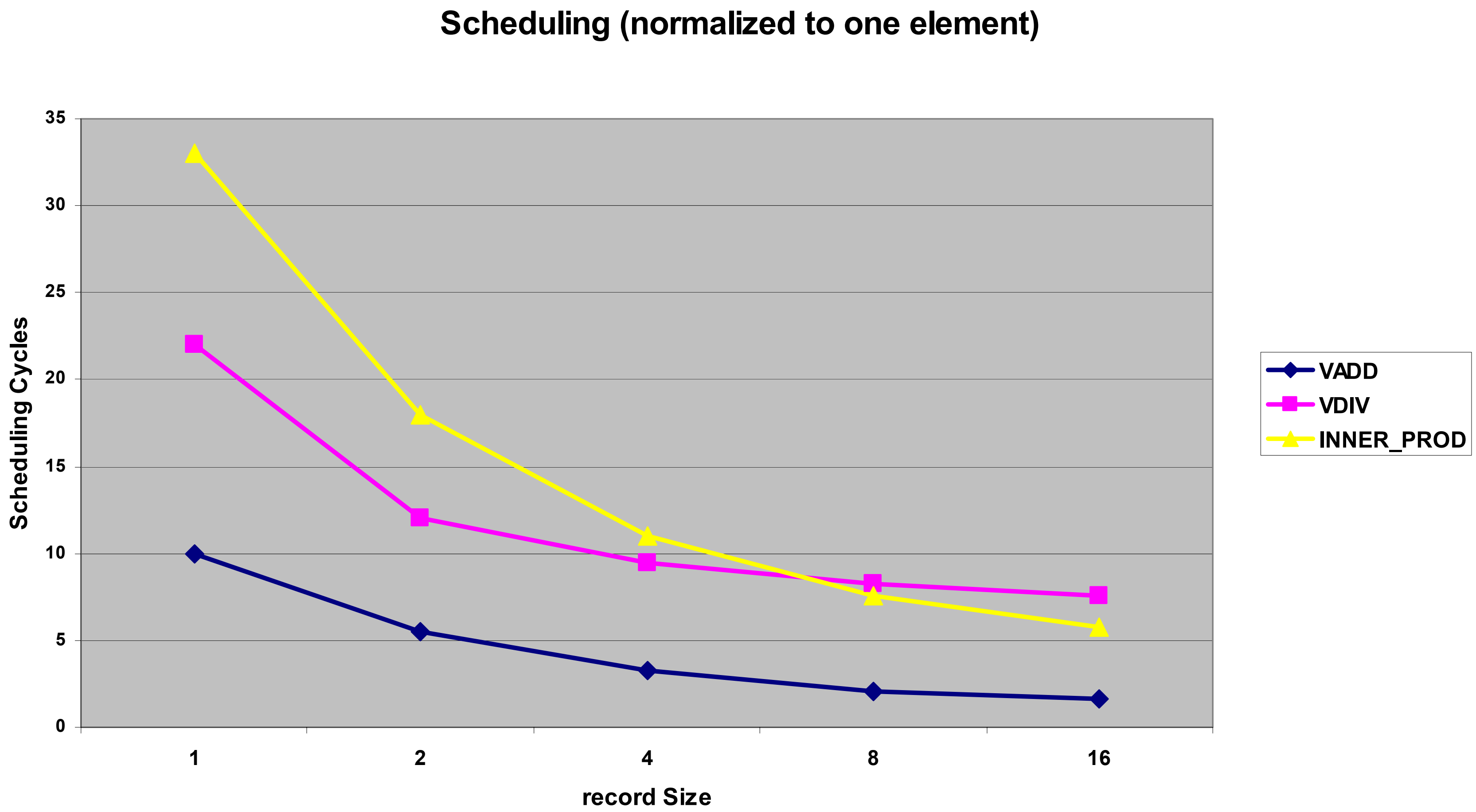
```
C[15:0] = A[15:0]  + B[15:0]

record vect {float v0, …, vn;}
kernel VADD(istream<vect> A, istream<vect> B, ostream<vect> C)
```

- Modulo Operation
  - Kernel granularity

```
C[15:0] = A[15:0]  + B[15:0]
E[15:0] = C[15:0]  . D[15:0]
kernel VADD(A,B,C)
kernel VMUL(C,D,E)              kernel VADD_MUL(A,B,D,E)
```

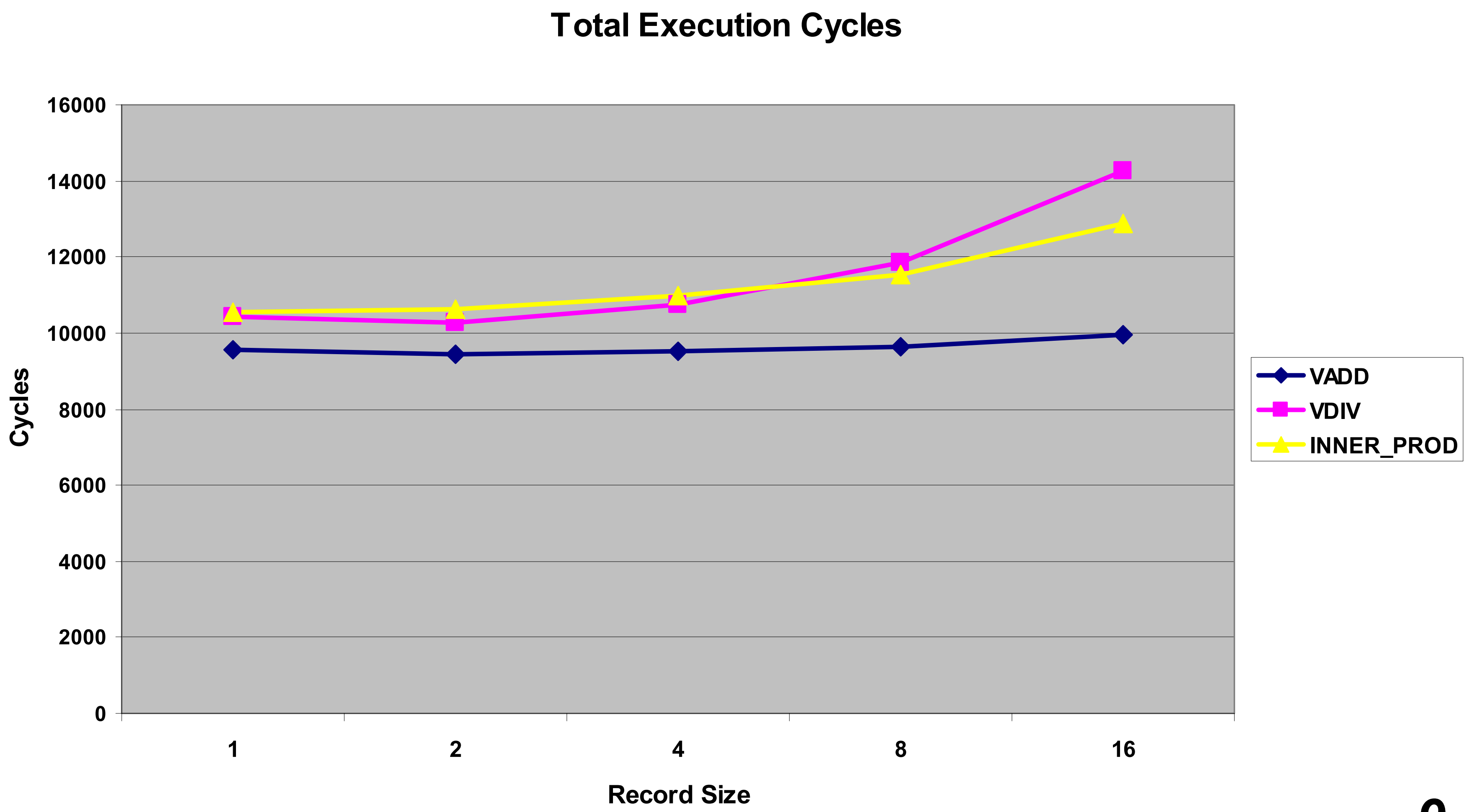
Effect of Record Size on Scheduling

- **Better scheduling with larger record sizes**
- **Unrolling has the same effect of increasing record size**



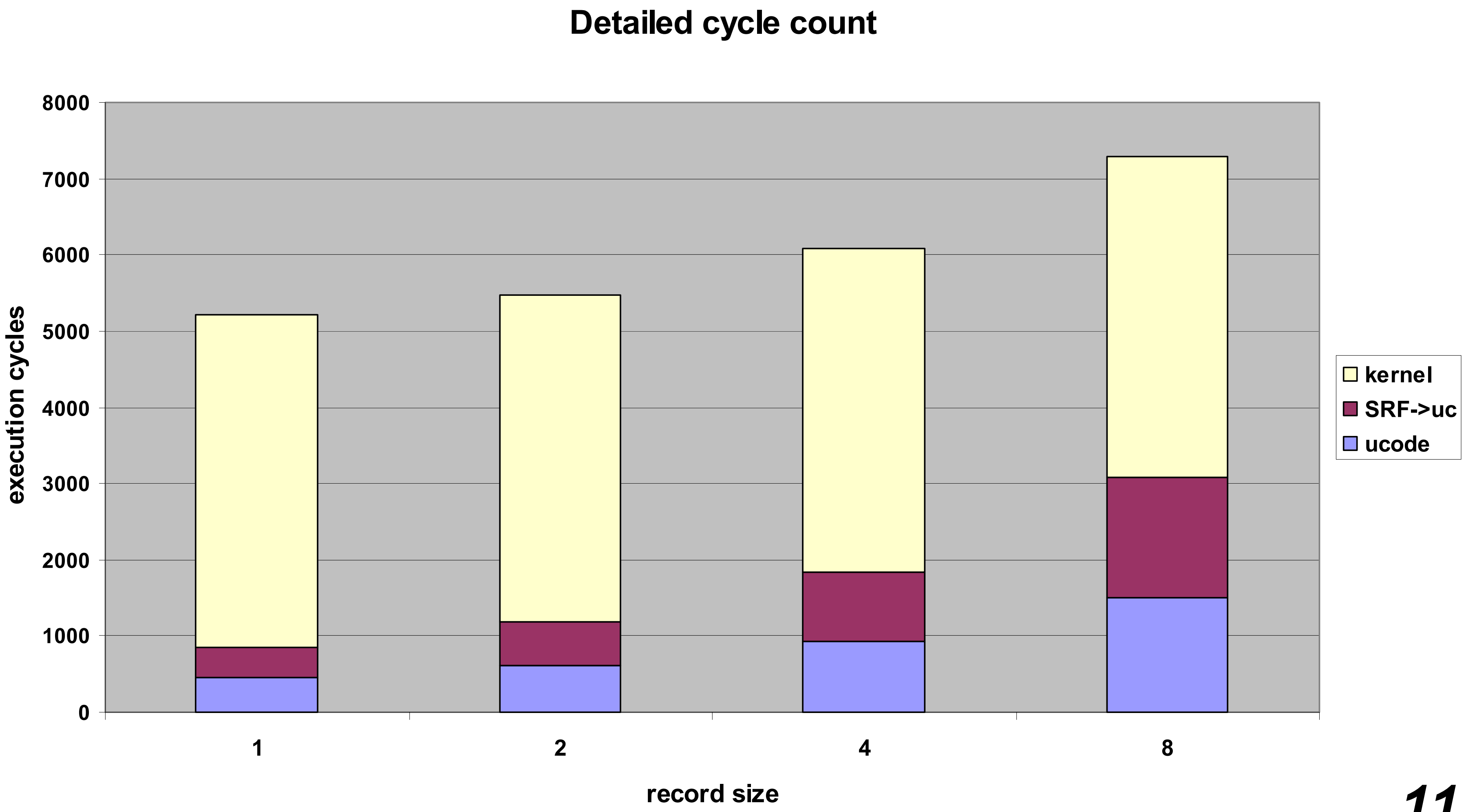
Total Execution Time

- **Not as expected!!!**



Why Worse?

Reason

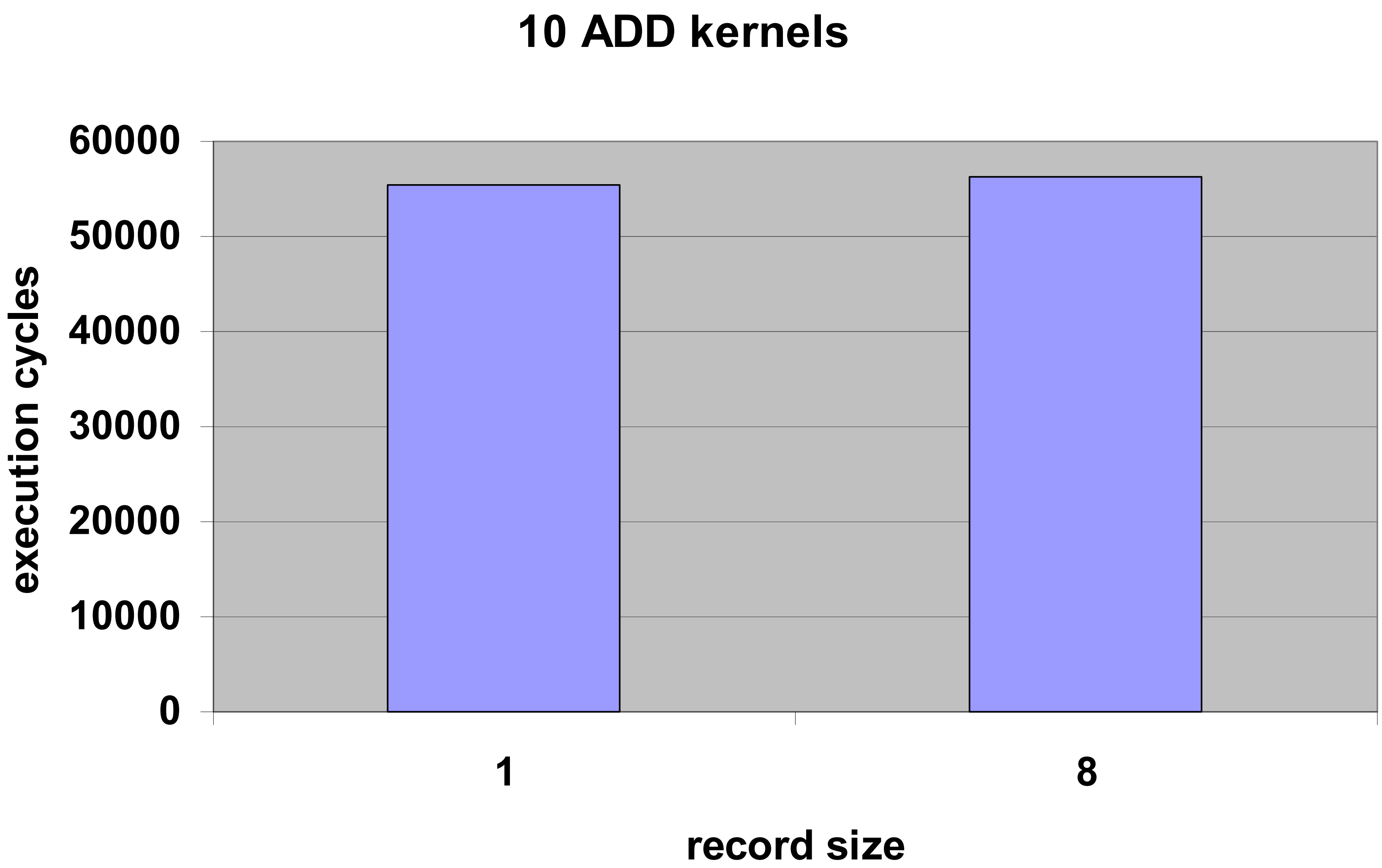


Detailed cycle count
8000
7000
6000
5000
4000
3000
2000
1000
0
execution cycles
1
2
4
8
record size
kernel
SRF->uc
ucode

μCode

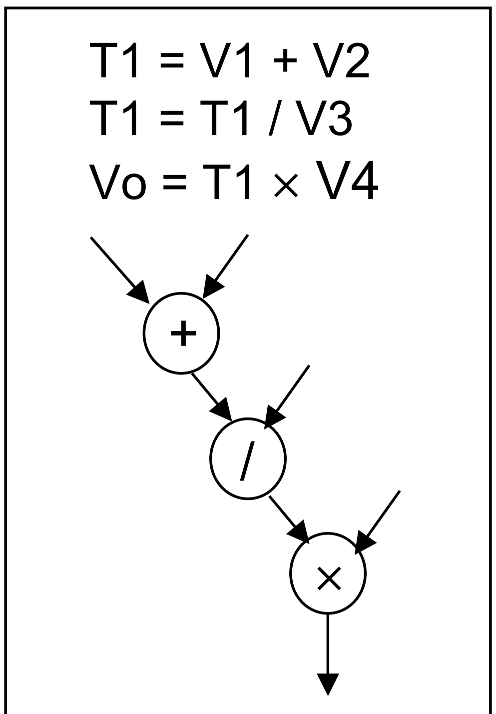
- μCode is first loaded to SRF
- Then loaded from SRF to μController
- Record size $\uparrow \Rightarrow$ μCode size $\uparrow$
- μCode cost can be *amortized*
  - reusing the same kernel
- Less of an issue for larger data sets

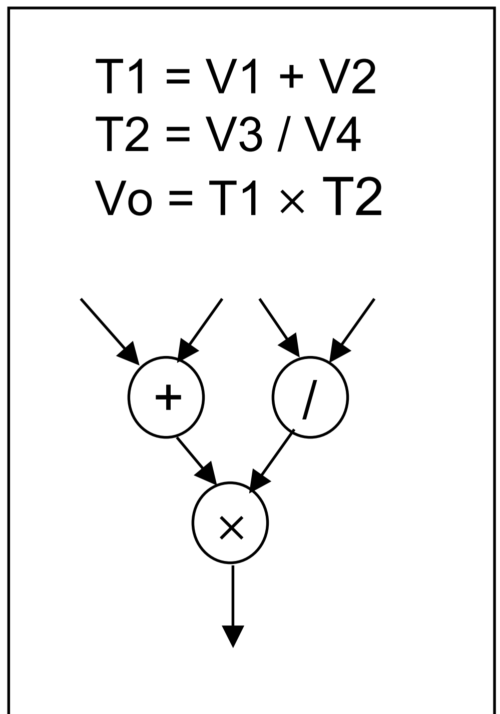
Amortized μCode

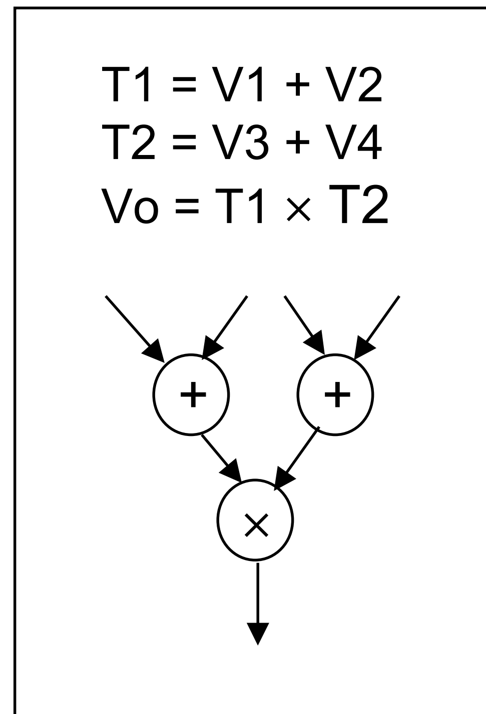


Kernel Granularity

- **Extreme cases:**
  - **Each operation in a separate kernel**
  - **All operations in one big kernel**

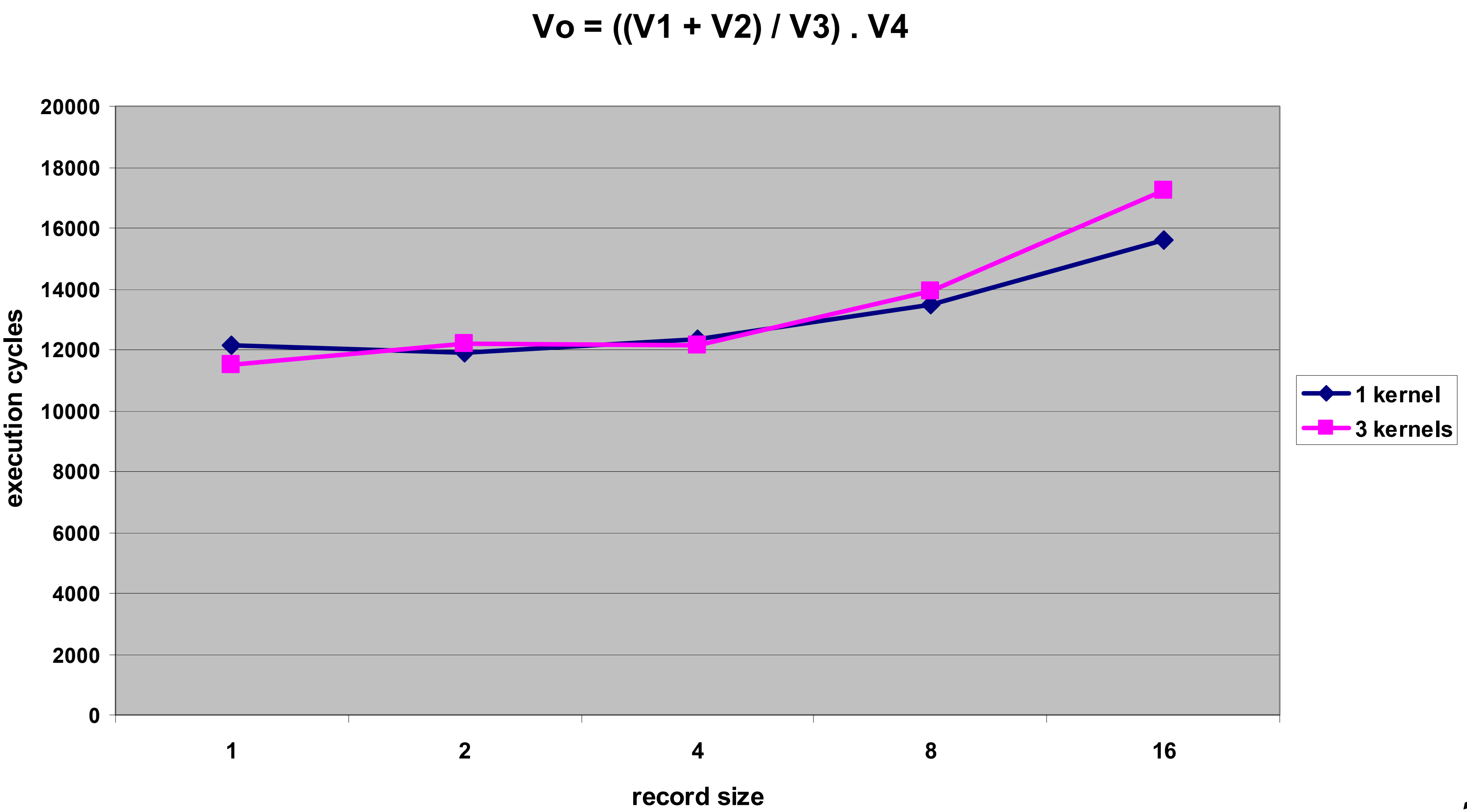






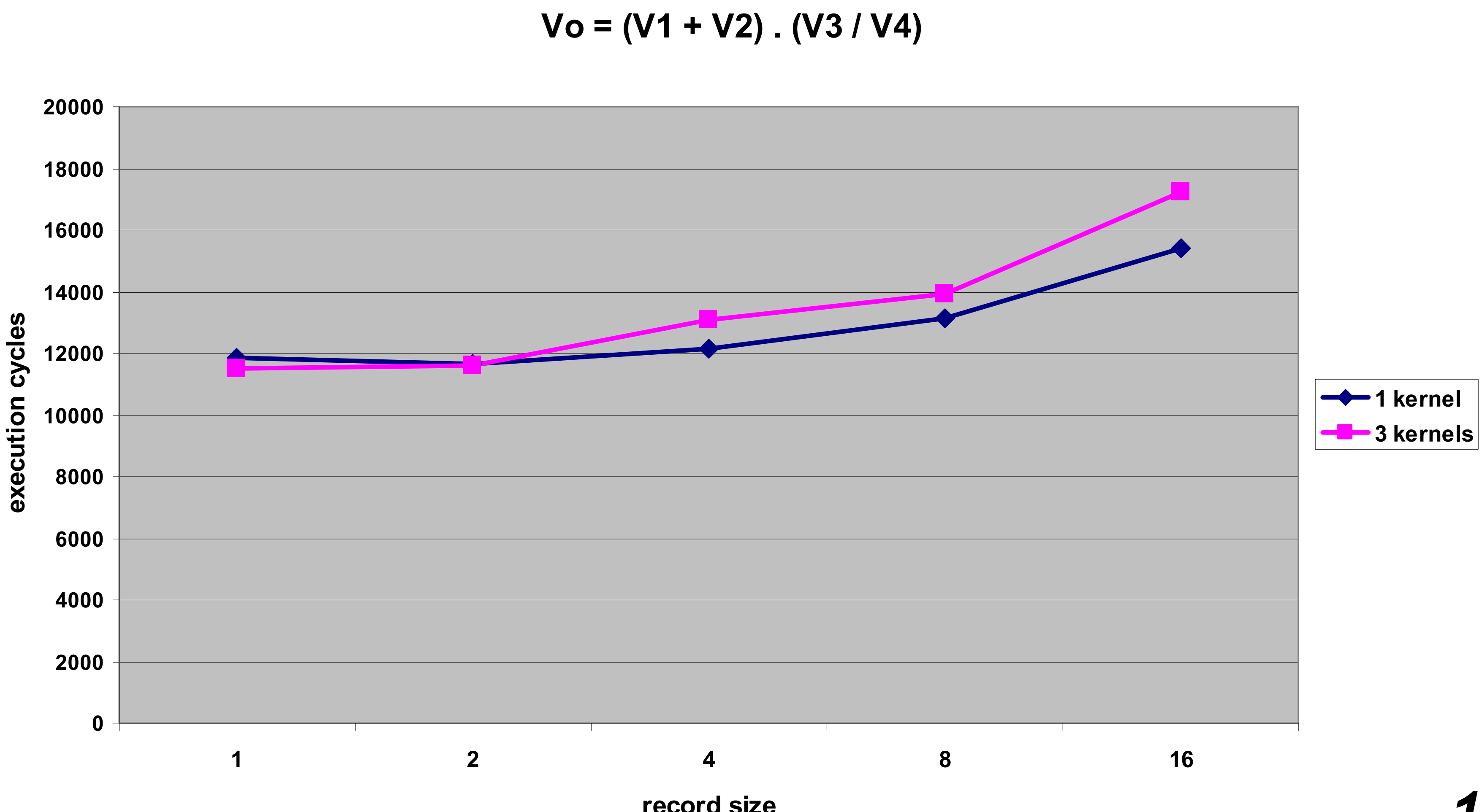
Serial Computations

- **256 data set**
- **No software pipelining in kernel scheduling**



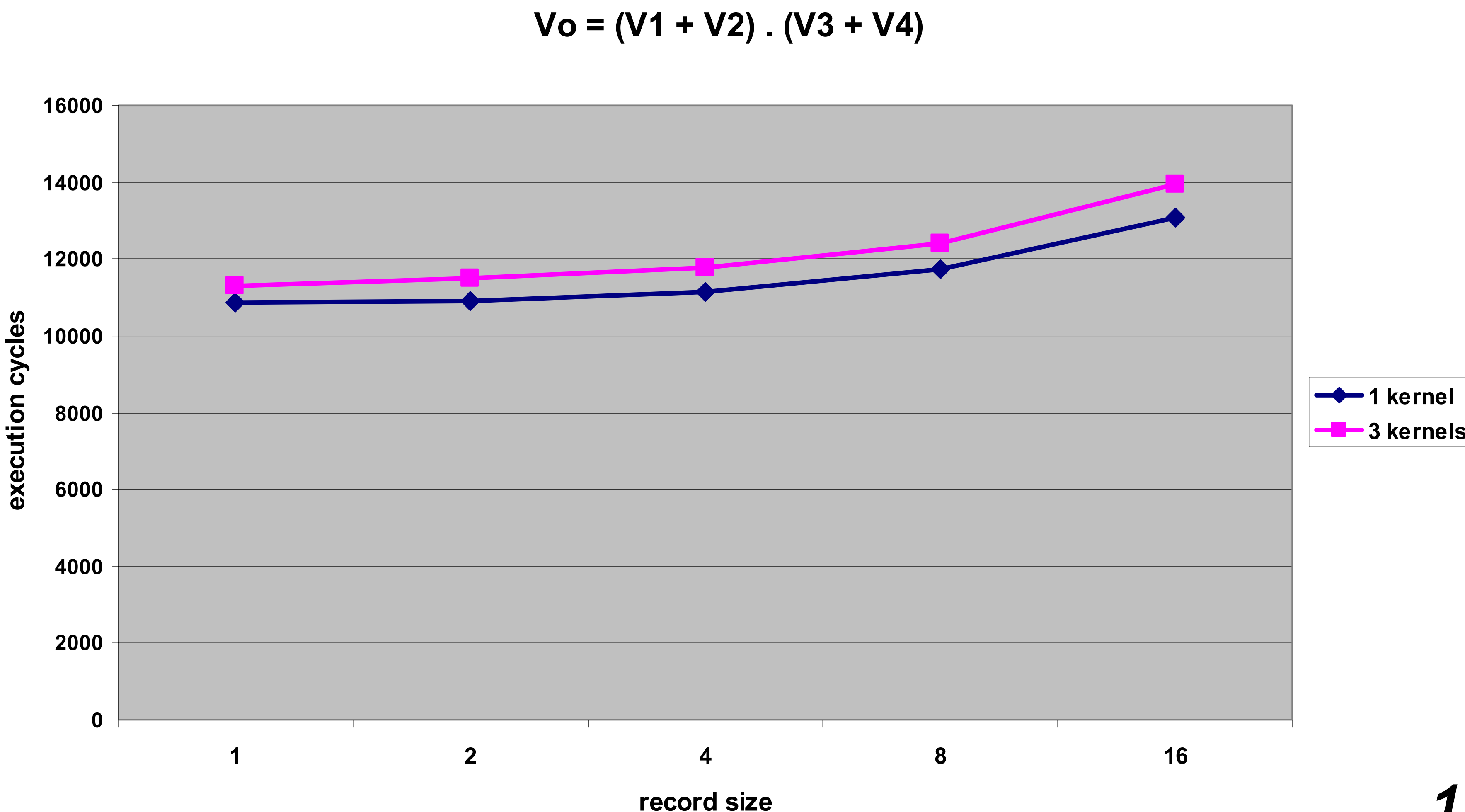
Non-serial Computations

- **256 data set**
- **No software pipelining in kernel scheduling**



More Non-serial Computations

- **256 data set**
- **No software pipelining in kernel scheduling**



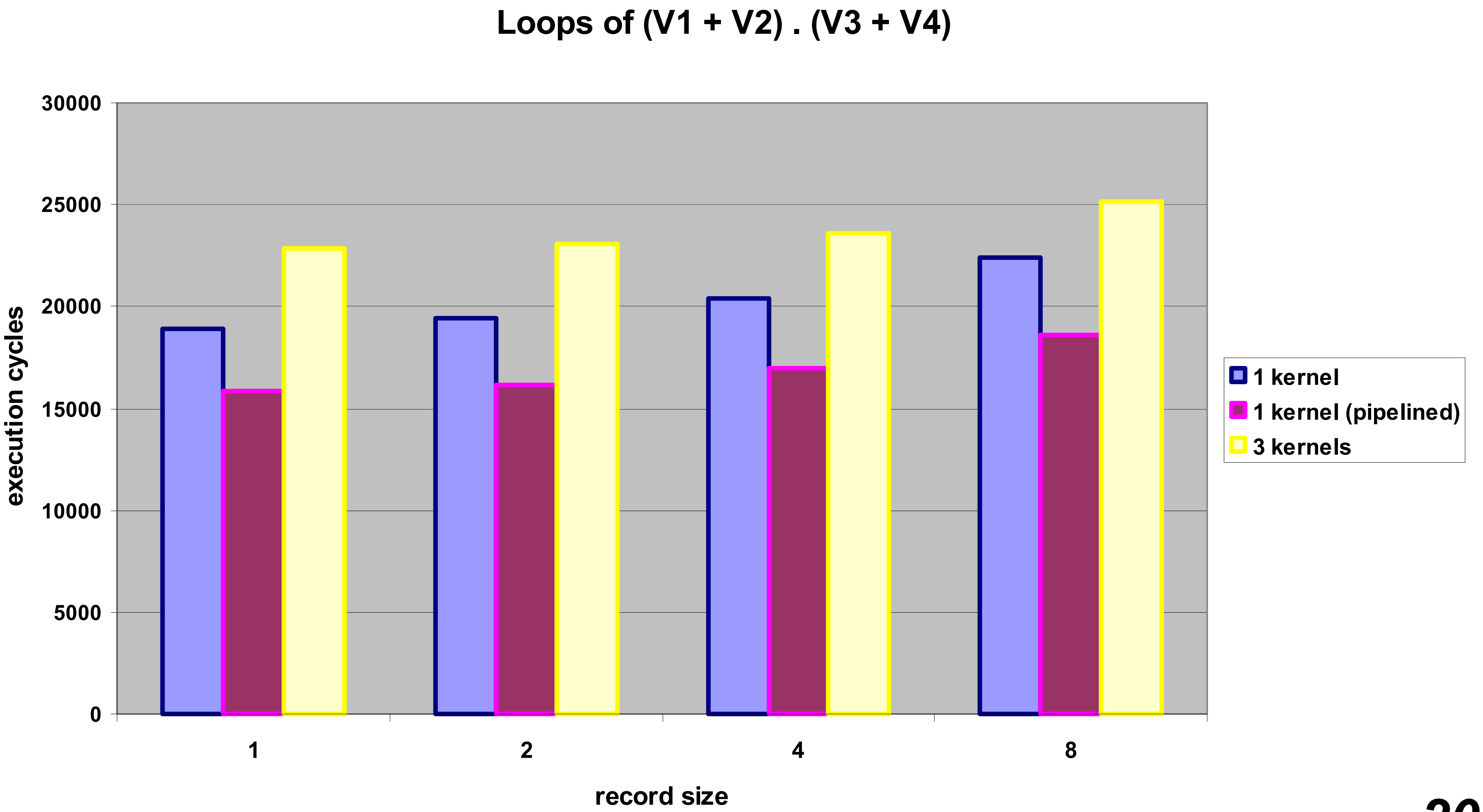
Two cases

- For serial computation
  - Smaller kernels with smaller record sizes
    - Best performance
- For non-serial (parallel) computation
  - Bigger kernels always better
    - Better resource utilization

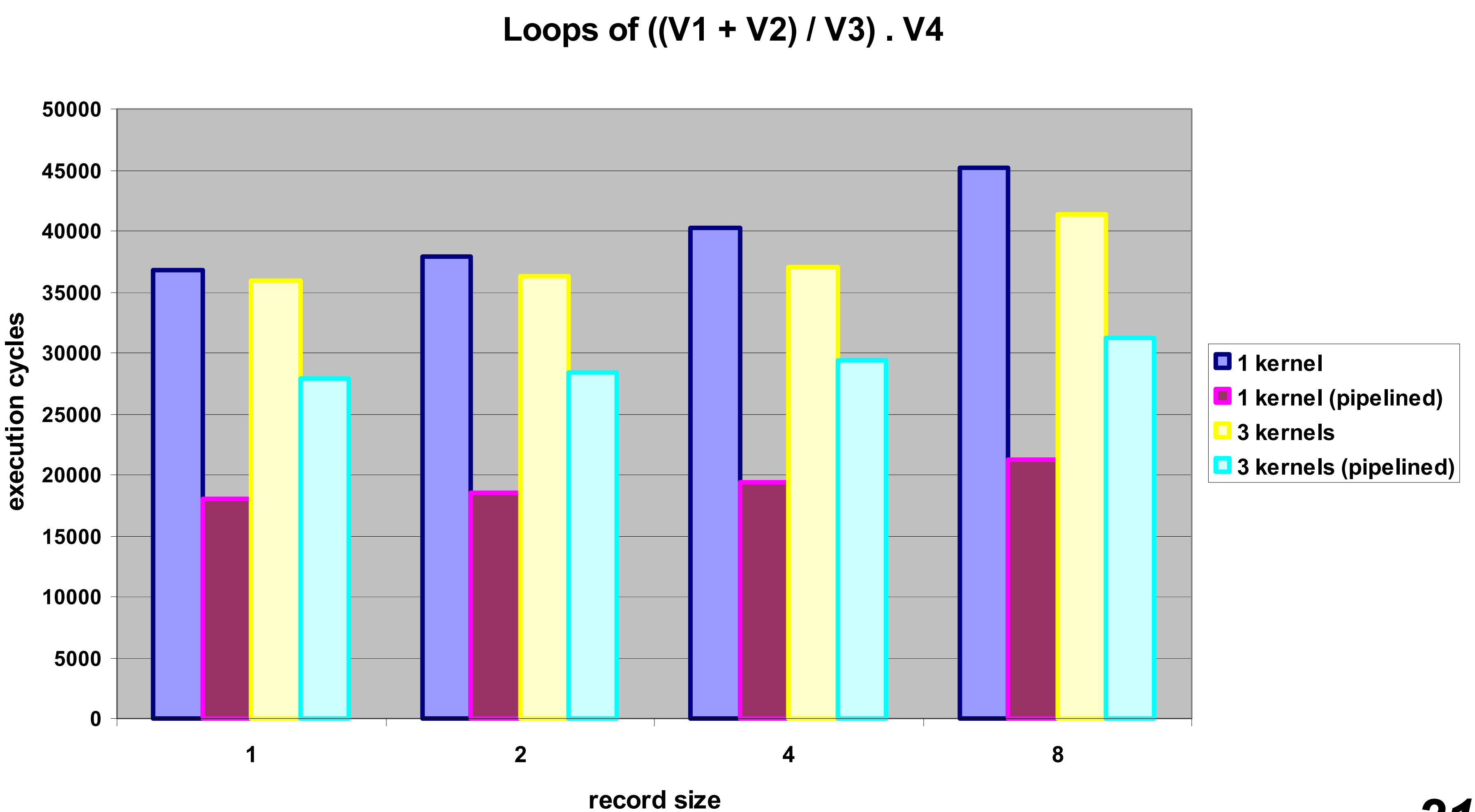
More Simulations

- Computational intensive operations
  - Heavy loops
    - Carry independent
      - Large Matrix by Vector manipulation
- Effect of software pipelining
  - Better resource utilization

Non-Serial Computation

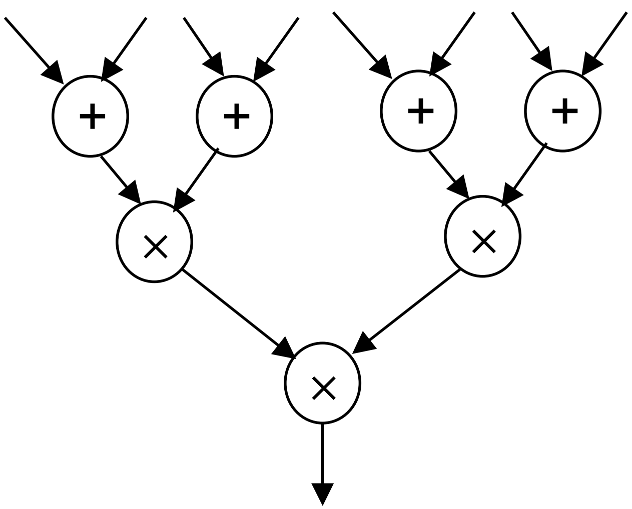


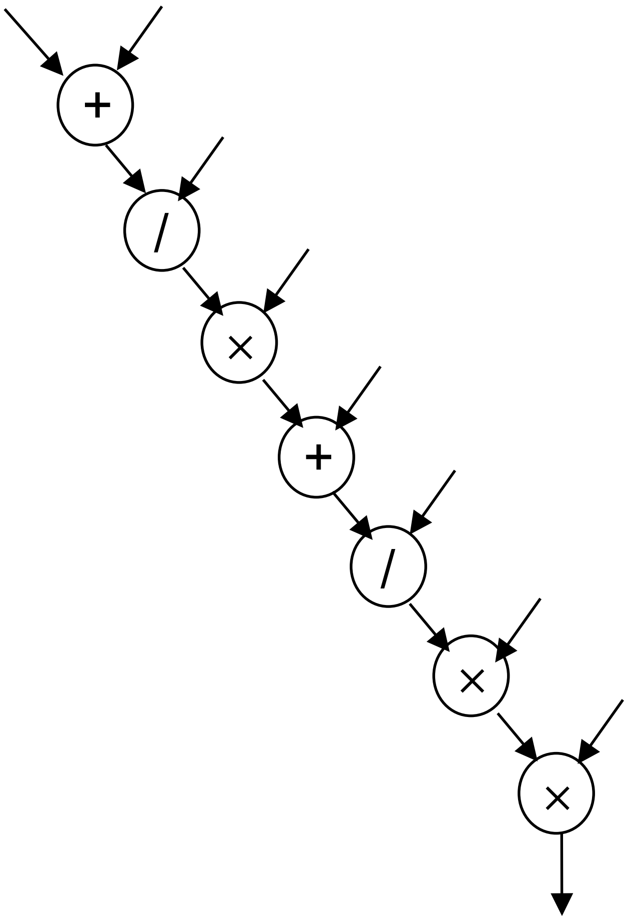
Serial Computation



Larger Case

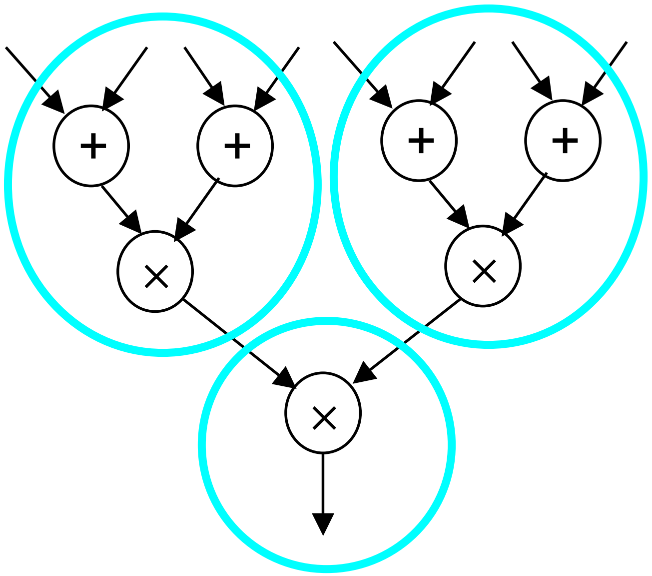
- Two Dataflows
  - Balanced tree
  - Fully dependent

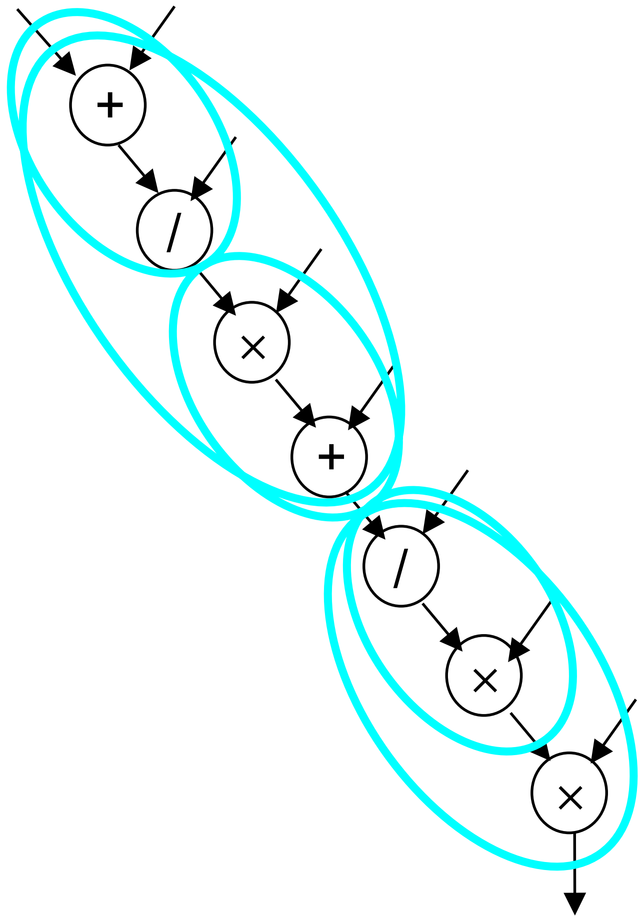




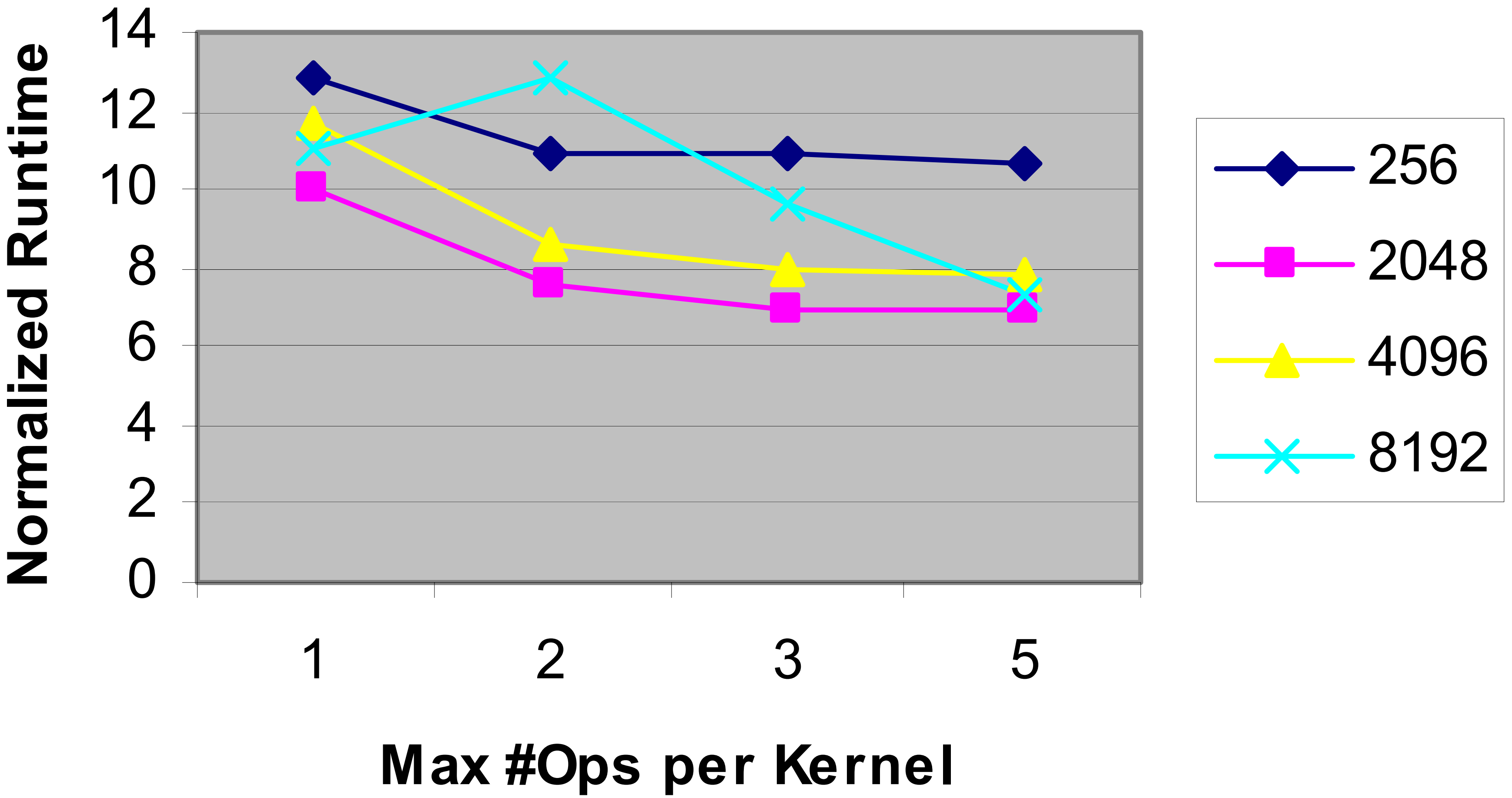
Kernel Fusion

- **Essentially kernel fusion**
  - **Merge 1-op kernels**

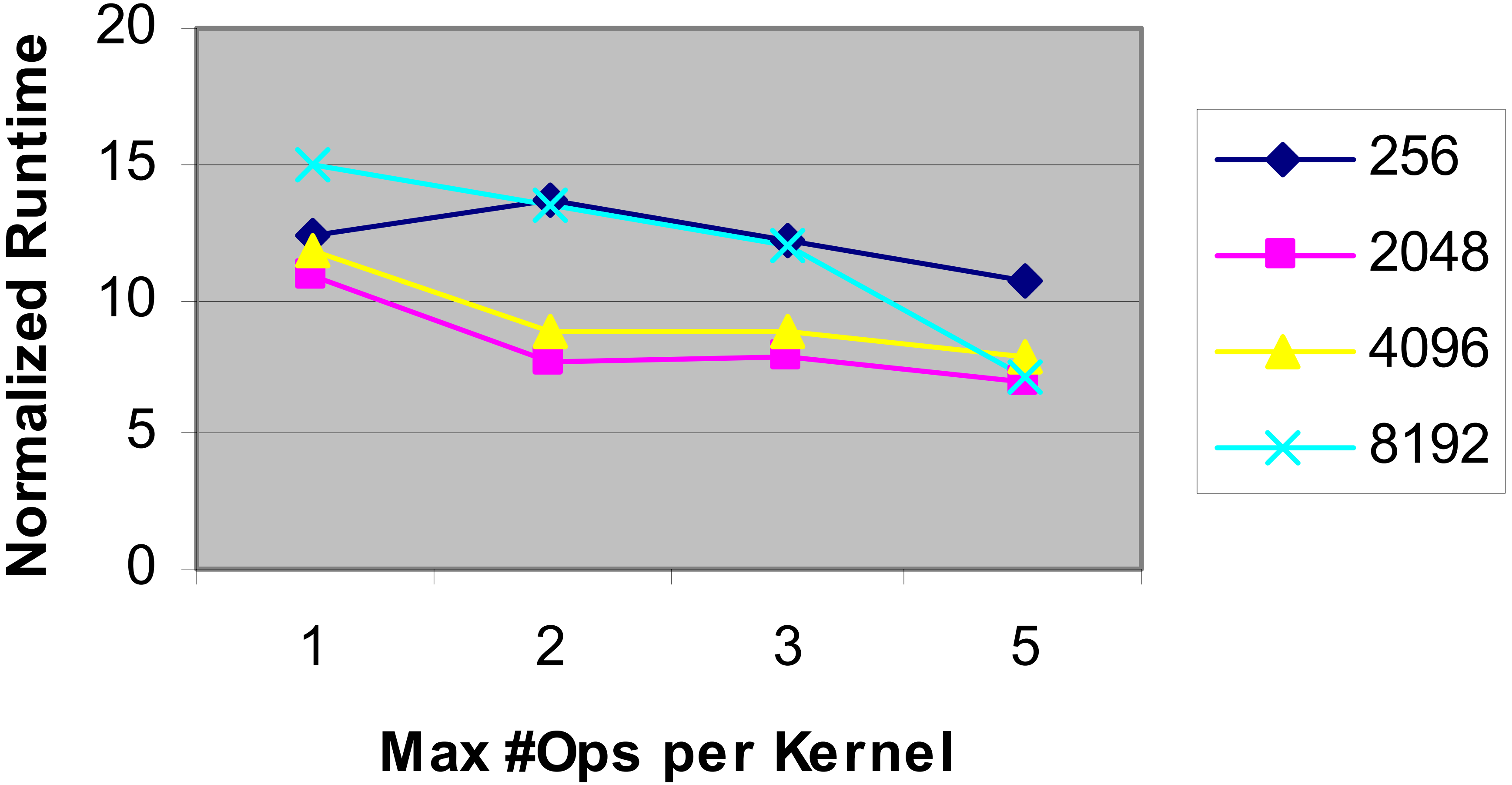




Serial Chain



Parallel Chain



Kernel size

- **Larger kernels are better for reasonable data size**
  - **More ops to schedule**
  - **Once there are enough ops, no more benefit**
  - **But, for data size comparable to SRF, large kernels still better**
- **Limits to kernel size**
  - **LRF size limit**
  - **Limit to number of streams per kernel**

Conclusion

- Explored basic issues of mapping vector code to stream code
  - Mostly confirmed intuition
  - Found a few issues we did not consider

- Next logical step : set of criteria for kernel fusion
  - Need to satisfy many constraints
  - Best solution may be impractical to find
  - Set of heuristics would probably suffice