A Near-Memory Processor (NMP) for Vector, Streaming, and Bit Manipulation Workloads

Mingliang Wei, Marc Snir, Josep Torrellas, and R. Brett Tremaine

University of Illinois and IBM

February 2005

Motivation

- Memory system bandwidth is expensive

- Many scientific/multimedia apps
  - Are memory-bandwidth limited (caches hardly work)
  - Could use vectors
  - Manipulate bits

- Commodity processors not optimized for this!

Approach

- Previous talk: NMP for prefetching
- This design: NMP to off-load computation from main proc
  - Vector
  - Streaming
  - Bit Manipulation
- Design is very simple
- **Near Memory Processor (NMP)**:
  - Optimized for bandwidth (e.g. no caches)
  - Not necessarily physically closer to memory

Near Memory Processor

- NMP: Simple multithreaded proc with vector/stream/bit support
  - High performance: tolerates high variability in:
    - Load/store latency of vector elements
    - Relative speed of communicating streams
  - Simple, compact design
  - Easy to program
  - Fairly general purpose
  - Can be on the main processor chip or closer to memory

- Uses scratchpad: flexible memory area to store stream buffers, vectors, and other

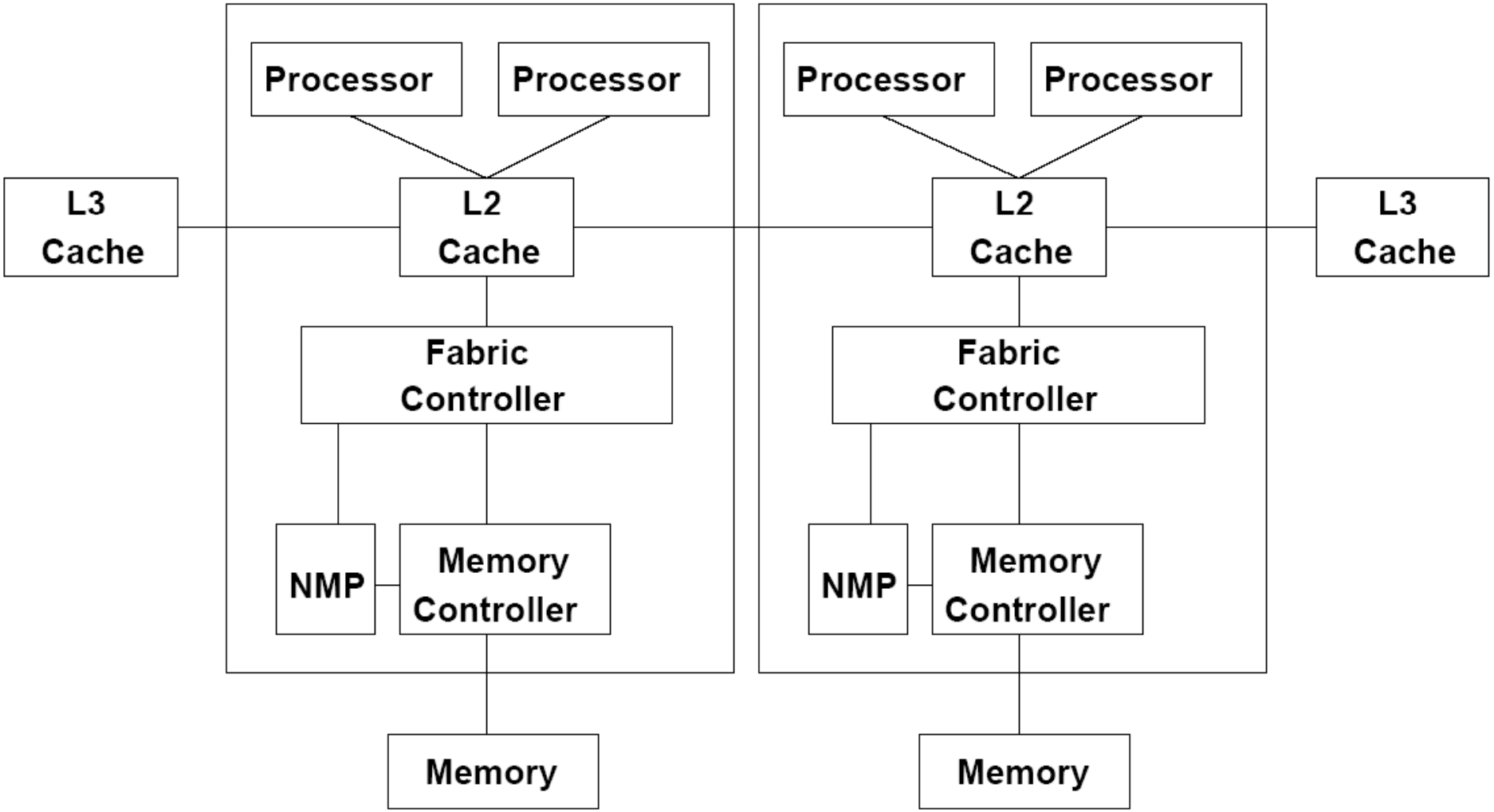
Vectors, Streams and Multithreading

- Vectors
  - ⇧ Have parallelism
  - ⇩ Caches work sub-optimally
- Streams
  - ⇧ Have parallelism
  - ⇧ Exploit producer/consumer
  - ⇩ Space multiplexing the hardware is inefficient
- Blocked Multithreading
  - ⇧ Tolerates variability in LD/ST latency of vector elements
  - ⇧ Tolerates variability in relative speed of communicating streams
  - ⇧ Simple implementation, efficient hardware use

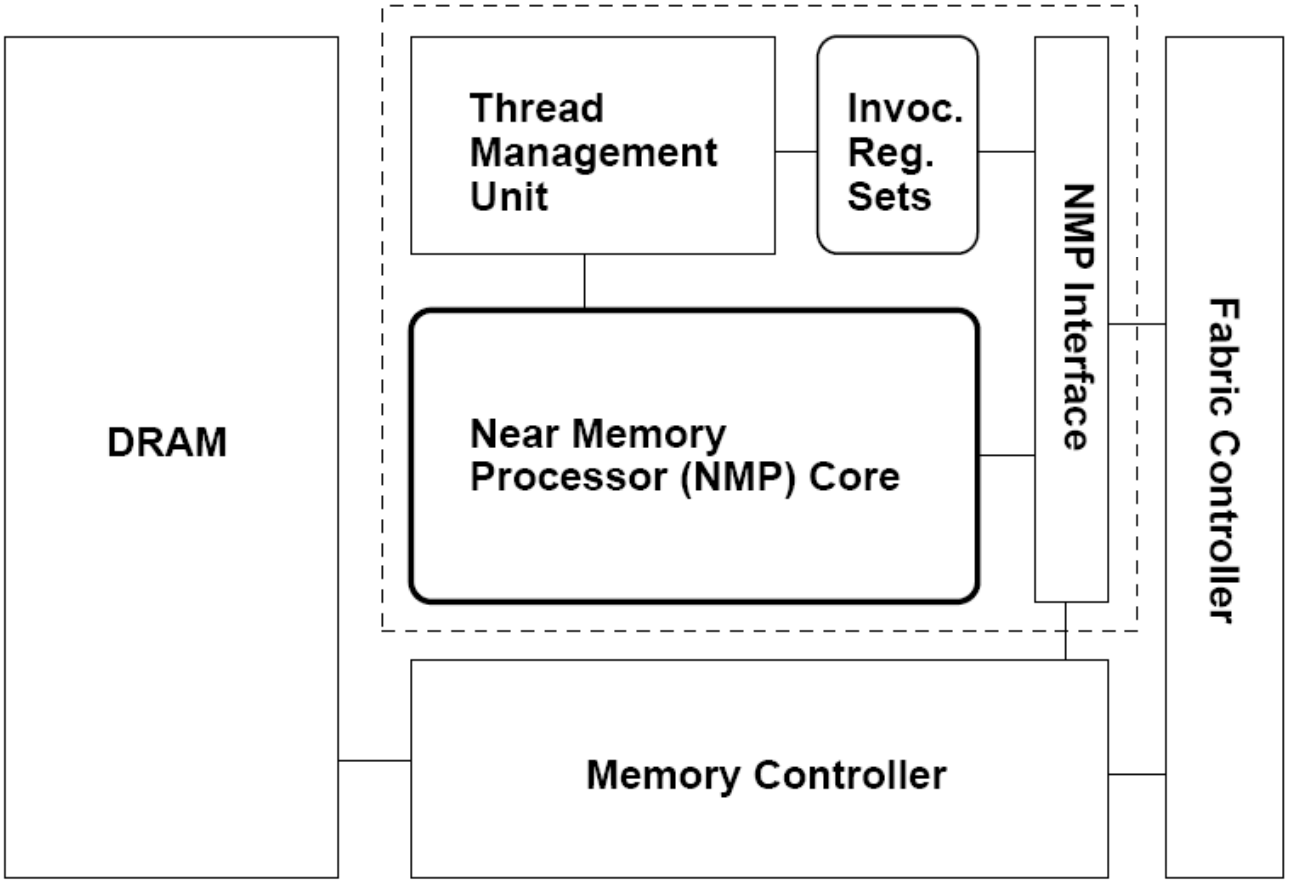
Multithreading in the NMP

- Context switch when processor expects long idle time
  - Scratchpad miss
  - Stalled on synch (e.g. fast producer stream kernel)
- Only a few contexts needed
- Do not save much state on context switch:
  - Scratchpad is not saved

System Architecture

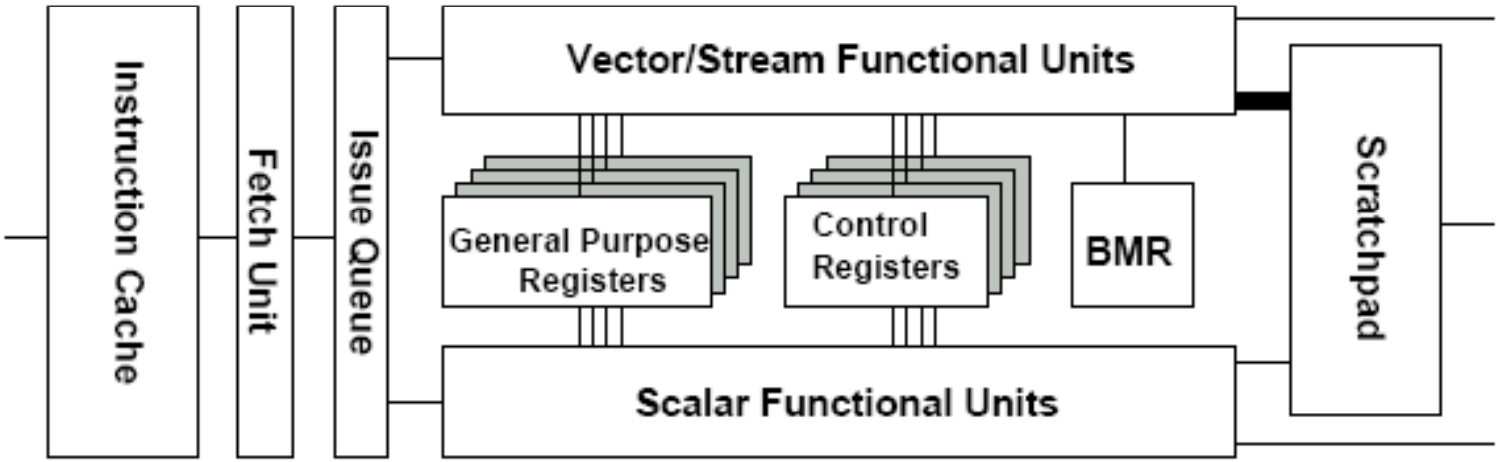


NMP Architecture



- In-order core

Core Architecture



Scratchpad

- High-bandwidth local memory
- Addressed with virtual addresses
- Shared by all threads running on the NMP
- Not saved on context switch
- Very efficient synchronization: Full/Empty bit per byte
- May suffer page misses
- Pages are lazily paged out to memory

Bit Manipulation

- Has Bit Matrix Register (BMR) of Cray
- Typical instructions:
  - Bmm_load: load 64x64 bit matrix
  - Bmm: bit multiply vector or scalar with the matrix in the BMR
  - Leadz
  - Popcnt
  - Sshift
  - Mix

Other Issues

- Interface with Main Processor (MP)
  - Start: MP stores into a mem-mapped location (no syscall)
  - End: NMP sets memory flag; MP polls

- Exceptions in NMP
  - Virtual memory exceptions:
    - Not precise but restartable and handled in SW
  - Each Scratchpad byte has a bit that records exceptions
  - Vector and stream buffers in Scratchpad cannot cross pages

Programming Environment

- NMP is attached to main program via system call

  NMP_handler = NMP_connect (ObjectAddress)

  Status = NMP_disconnect (NMP_handler)

- NMP is invoked from main program via asynchronous (user space) method invocation
  - Processor stores parameters at NMP "doorbell" and polls flag

    Status = Memthread_create (FunctionInvoked, Params, CompletionFlag, NMP_handler)

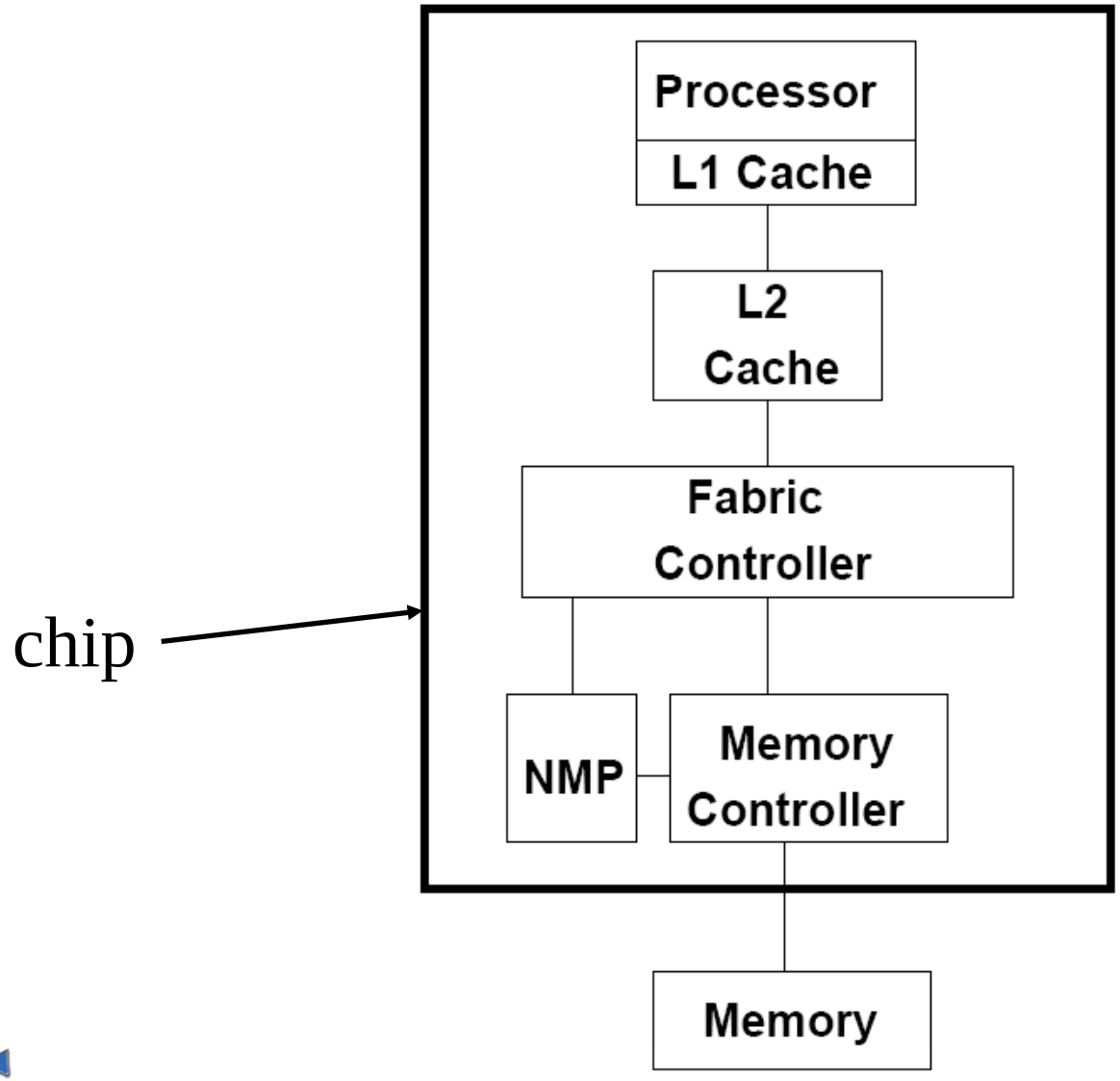
    Memthread_wait (CompletionFlag) or Memthread_poll (CompletionFlag)
  - NMP sets completion flag

    Memthead_end (CompletionFlag)
  - Fits X10 Asynchronous expressions and futures

Possible Programming Support

- Trivial: “hand-coded” libraries for special applications
- Easy: libraries separately compiled by vectorizing compiler
- Research: compiler that splits code into main-processor part and NMP part
  - Can leverage previous work at IBM and UIUC (FlexRAM)

Architecture Evaluated



Simulation Parameters

NMP Parameters	
Parameter	Value
Frequency	4GHz in-order
Issue Width	2
# Scalar FUs	1Int FU, 1FP FU
# Vector FUs	16Int FUs, 16FP FUs
# Lanes	16
# Pending Memory Ops (Ld, St).	128, 128
# Contexts	4
Time to Context Switch	4 cycles

Memory Parameters	
Parameter	Value
L1, L2, Scratchpad size	32KB, 1MB, 64KB
L1, L2 associativity	2-way, 4-way
L1, L2 line size	64B, 64B
Main proc. to L1, L2, memory round-trip latency	2, 10, 500 cycles
NMP to Scratchpad, memory latency	6, 470 cycles

Main Processor Parameters	
Parameter	Value
Frequency	4GHz out-of-order
Fetch Width	8
Issue Width	4
Retire Width	8
ROB size	152
I-window size	80
Int FUs	3
FP FUs	3
Mem FUs	3
Pending Ld/St	16, 16
Branch Pred.	Like Alpha 21464
Branch Penalty	14 cycles

Simulator Infrastructure

- Architecture simulator from UIUC (SESC)
- Models ooo processors and MP memory hierarchies
- 110 K lines of C++ source code

Simulating an R4400 at 150Mhz (SGI IP22):

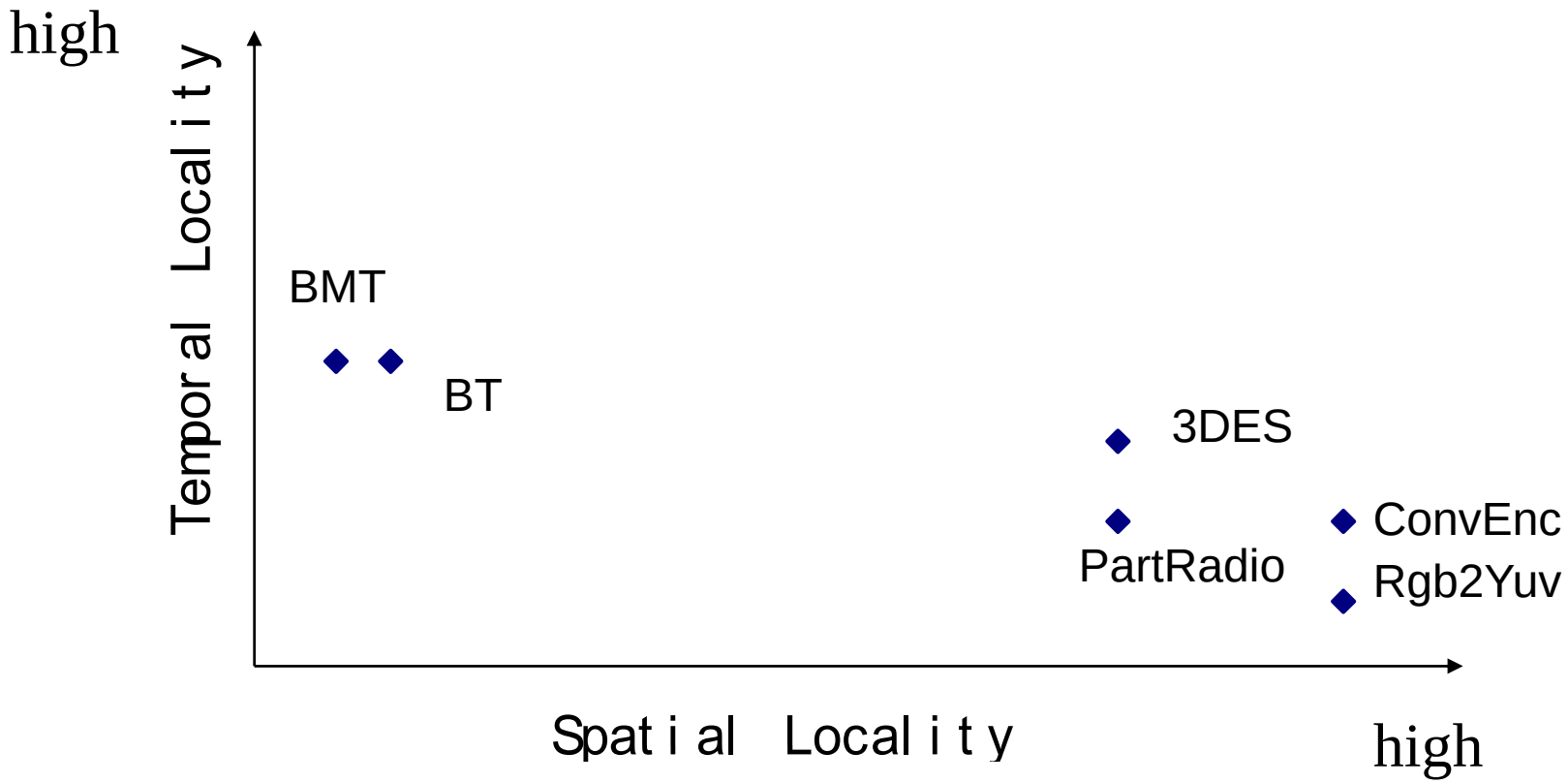
Bench	Native	Simulated	Error	SimTime
matrix	291ms	282ms	-4%	27s
crafty	265ms	271ms	+4%	43s
mcf	2438ms	2422ms	-1%	174s
mp3dec	2347ms	2521ms	+7%	185s
lat_mem	4005ms	4129ms	+3%	279s

Applications

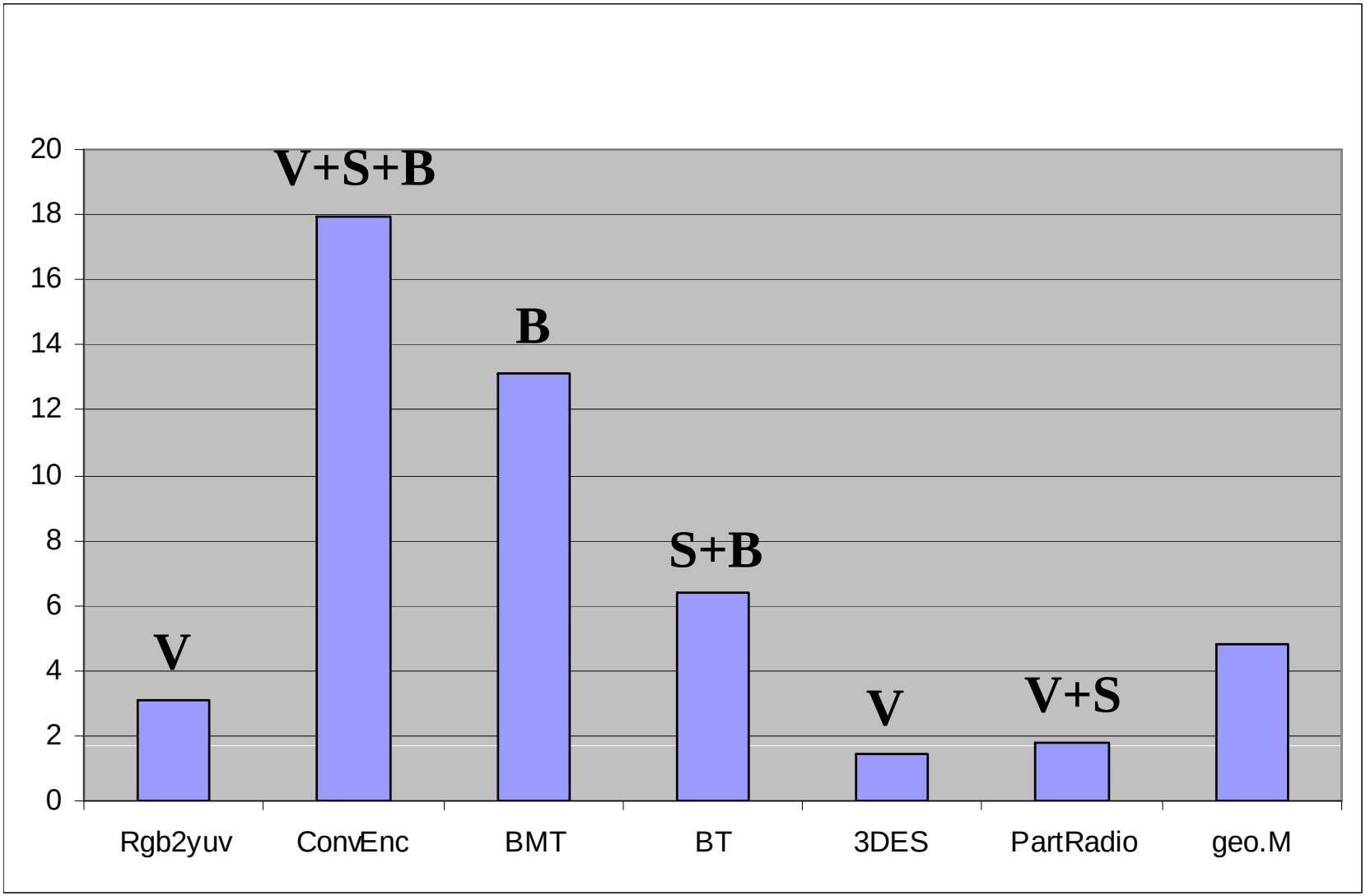
Name	Vec	Stm	Bit	#Thd	Remarks
Rgb2yuv	X			4	Convert the RGB presentation to YUV
ConvEnc	X	X	X	3	Convolutional encoder
BMT			X	4	Bit matrix transposition
BT		X	X	3	Bit twiddle
3DES	X			4	3DES encryption
PartRadio	X	X		3	Partial radio station

- Code length: 730 lines/app (average)

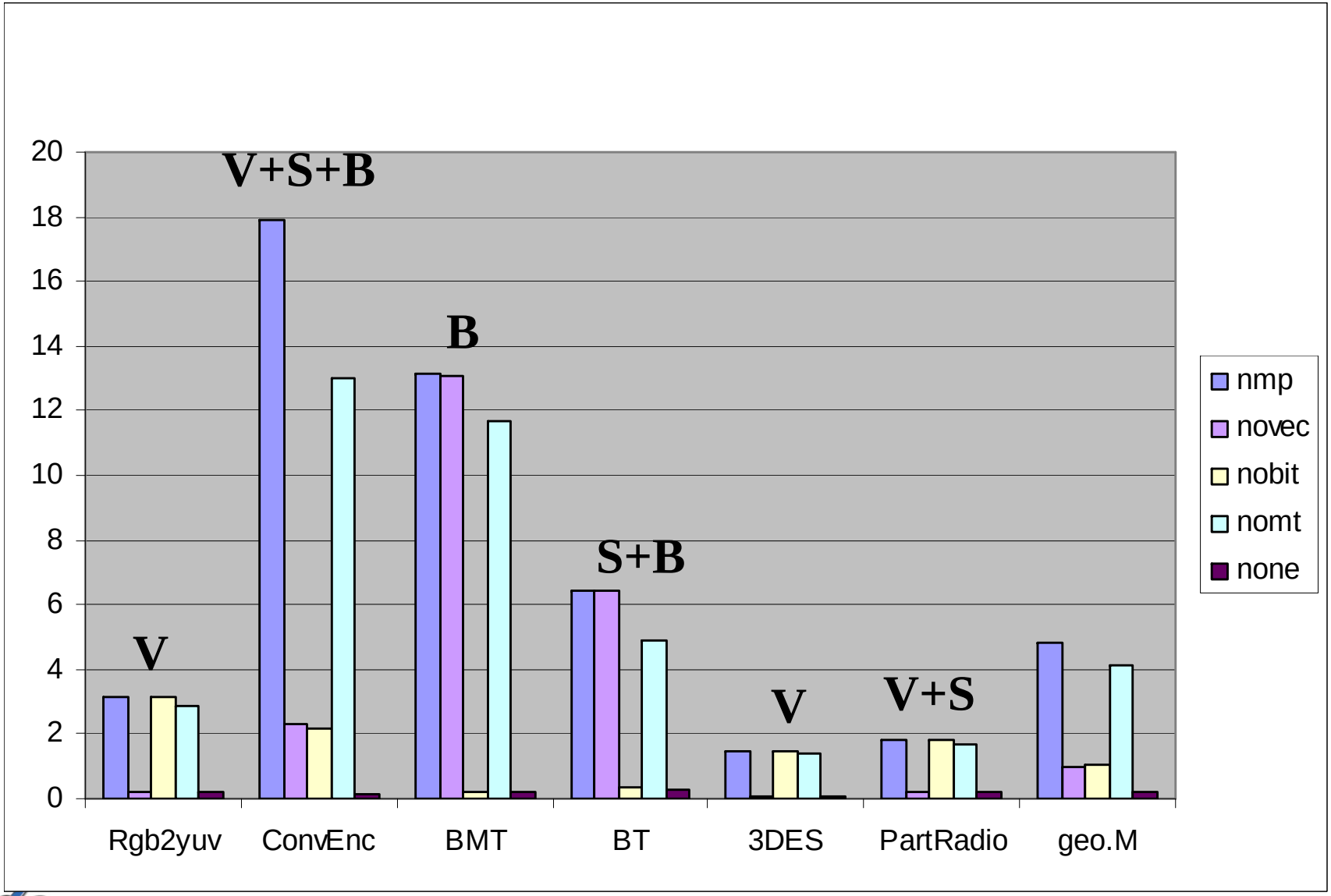
Mapping of Applications



NMP Speedup Over Main Processor



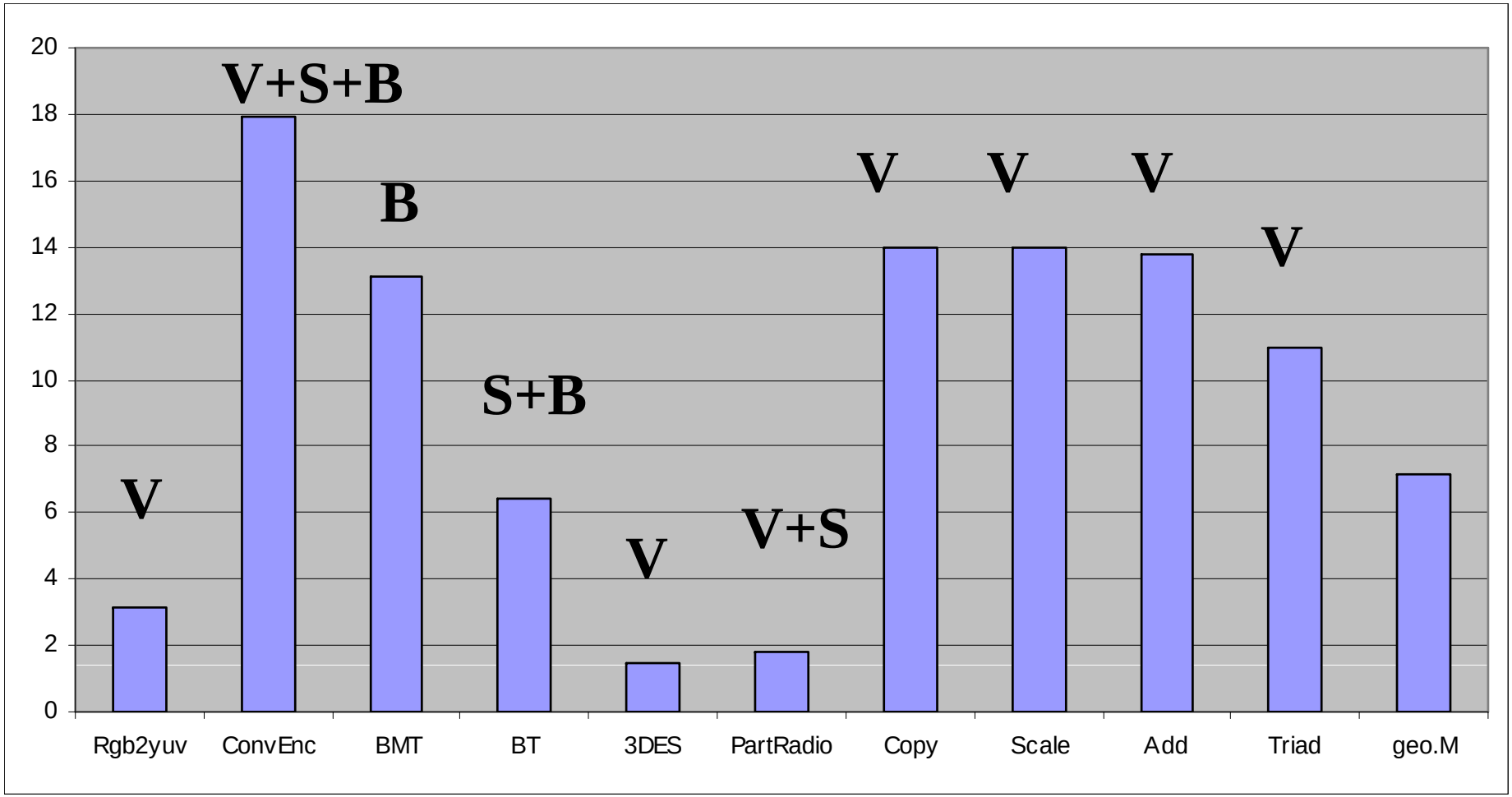
NMP Speedup Over Main Processor



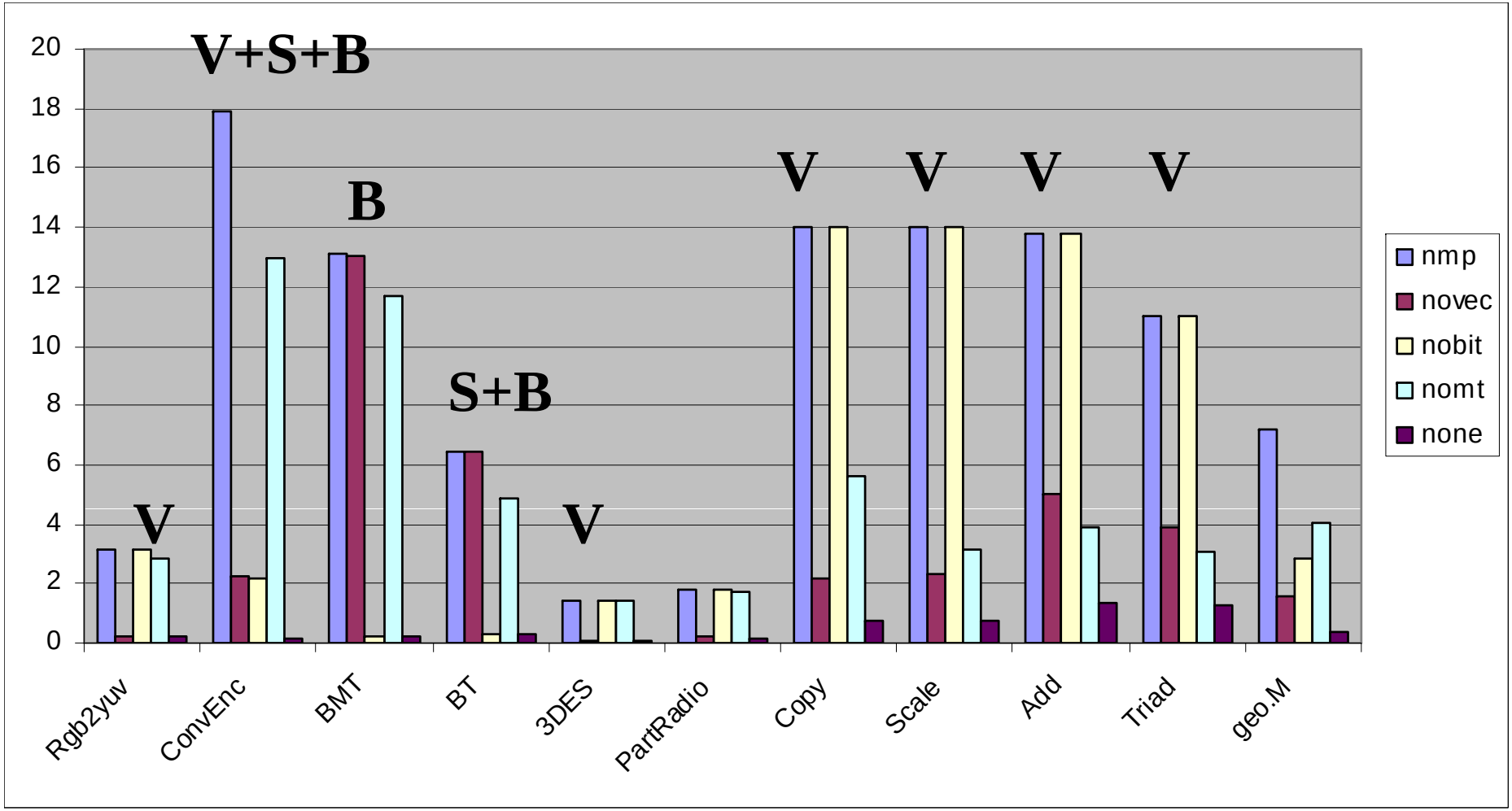
Results

- Nmp much faster than main: geometric mean speedup of 4.9
- Vector support is key to the speedups in many vectorizable apps (Rgb2yuv, 3DES, and PartRadio)
- Bit manipulation support is key in some apps (BMT and BT)
- ConvEnc needs both vectorization and bit manipulation support

Adding the Stream App



Adding the Stream App



Conclusions

- Selected one NMP design point:
  - Off-load vector, streaming, bit manipulation ops
  - Simple/compact design
  - Fairly general purpose core
- Evaluated design assuming NMP on processor chip
- High performance for high-bandwidth applications:
  - 4.9 x faster than main processor
  - Most cost-effective support in the NMP:
    - Vectors
    - Bit manipulation

Future Work

- Focusing on programmability:
  - Programming interface
  - Compiler support
- Interaction with the OS
- Step back: How far can we go with PowerPC ISA (and small extensions)?
  - No vector compiler
  - Libraries that perform efficient bit manipulation

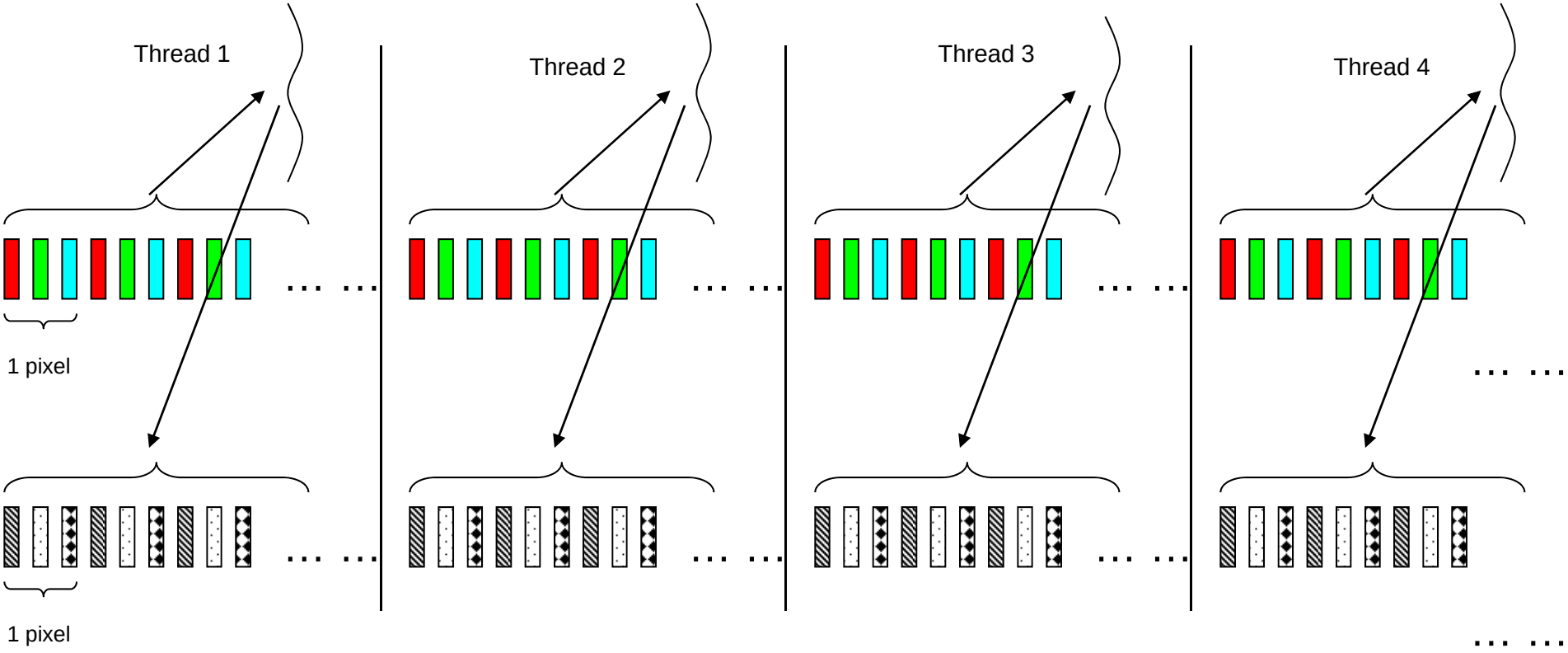
Near Memory Processing (NMP)

Mingliang Wei, Marc Snir, Josep Torrellas and R. Brett Tremaine

University of Illinois and IBM

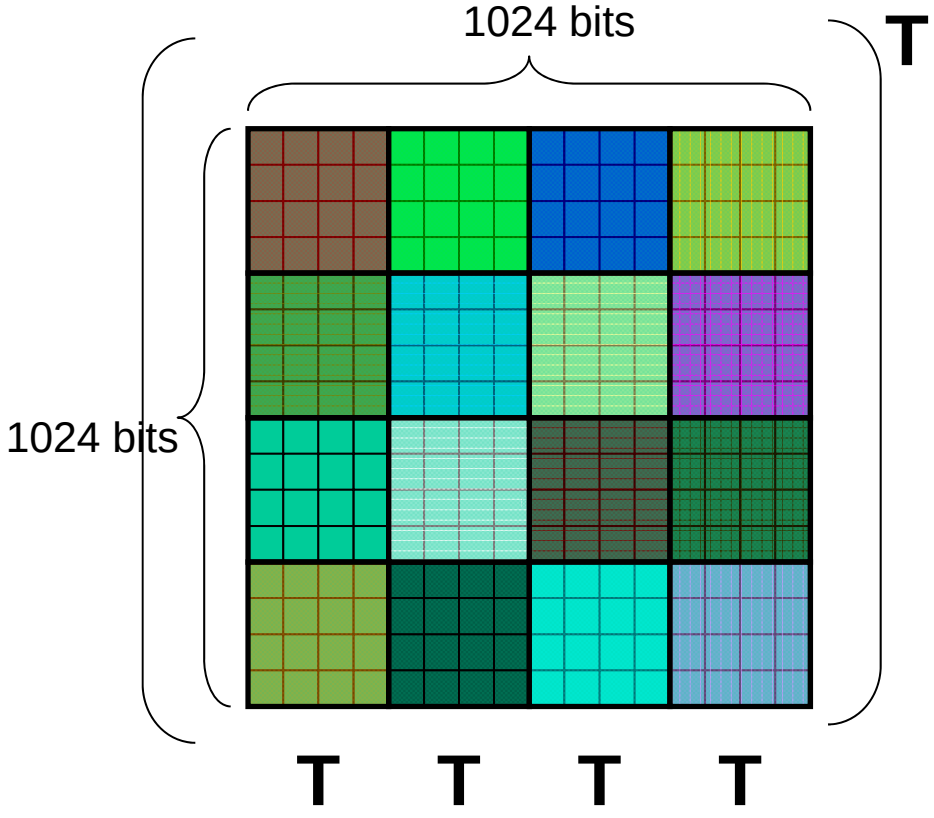
February 2005

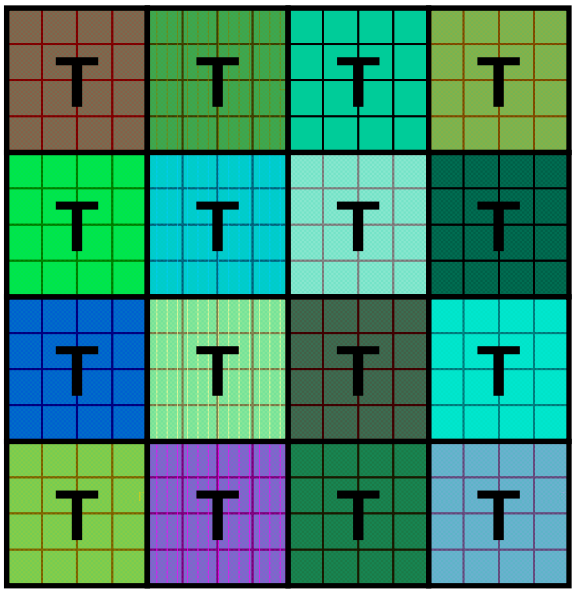
RGB2YUV (Vector)



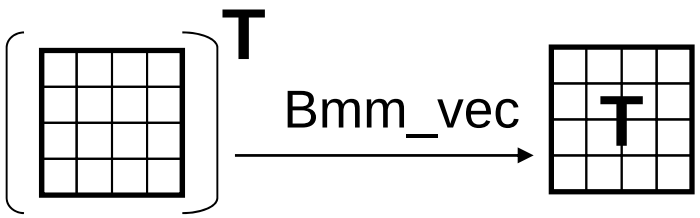
- Input: 1000 x 200 pixels, output is the same size
- Conversion of each pixel is independent
- 4 threads, each processes a partition of the input
- Each thread processes 16 pixels in parallel

BMT (Bit manip)





- Transpose 4 1024x1024 bit matrices
- 4 threads, each trans. 1 matrix
- Threads use:
  - Vector load to load data
  - Bmm_vec to trans. a tile
  - Vector store

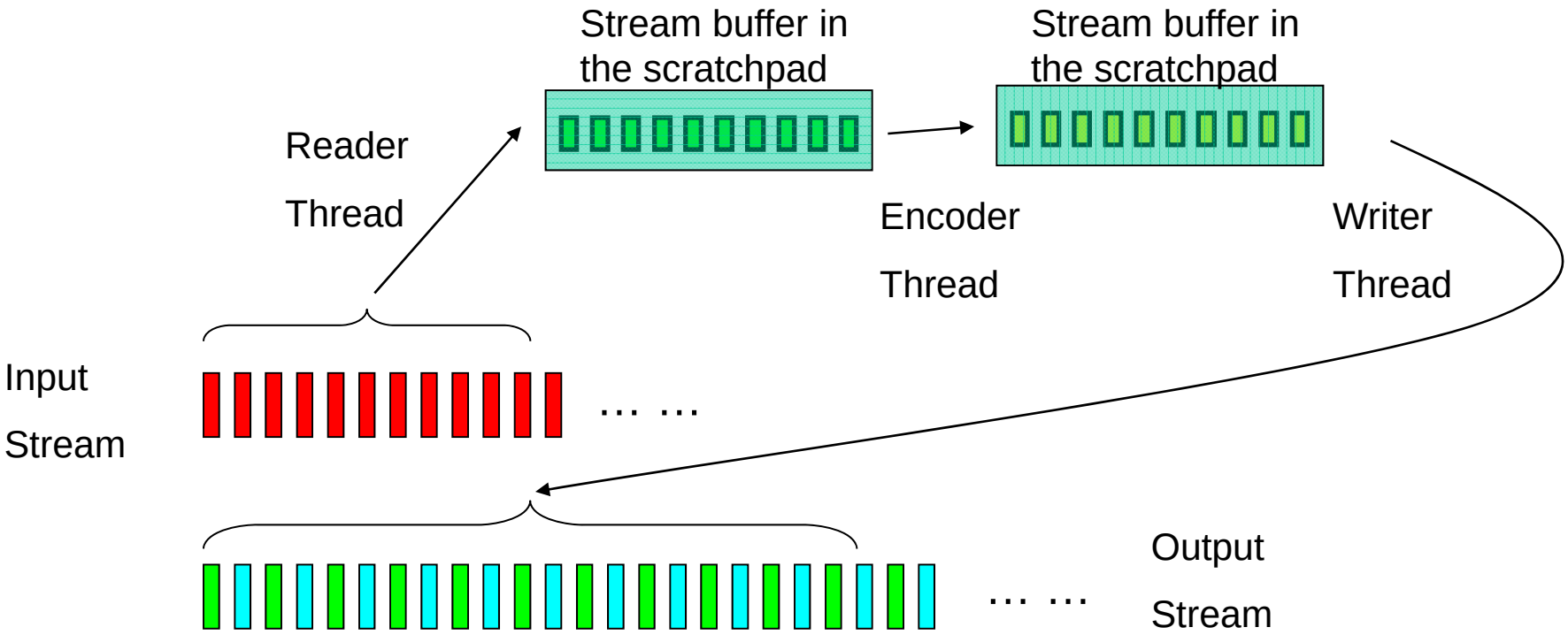


A 64x64 bit tile

3DES (Vector)

- Input: set of 64-bit word values. Output is same-size encrypted word values.
- Vectorized the load/store and the computation
- Counter mode (more parallelism)
- 4 threads, each work on a partition of the input (like Rgb2yuv)
- Each thread processes 16 elements in parallel
- Bmm to perform initial and final permutation (though a small fraction of total execution time)

ConvEnc (Vector + stream + bit)

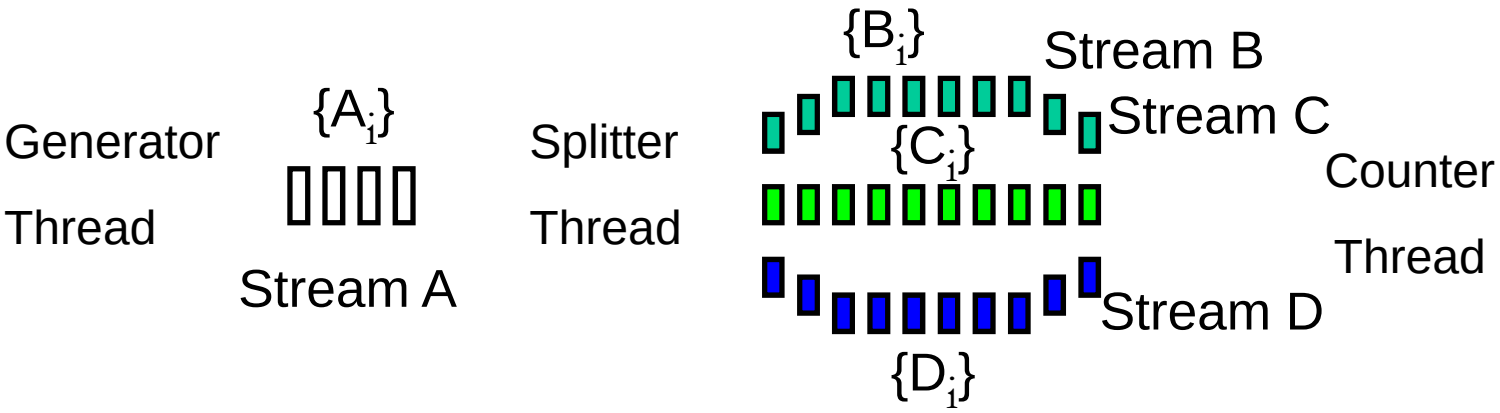


- Input: bit stream; output: 2x bit stream
- Encode 64 bytes at a time
- Sshift instruction to shift 64 bytes of data
- Mix instruction to interleave two blocks of bits

BT (Stream + bit manip)

Generate $\{A_i\}$
$\{B_i\}$ = take 5, drop 6, take 5 … from $\{A_i\}$
$\{C_i\}$ = drop 5, take 5, drop 6, take 5, drop 6 from $\{A_i\}$
$\{D_i\}$ = (C_i or not(B_{i+1})) xor (not (C_i) or B_{i-1})
$\{E_i\}$ = D_i xor D_{i+37} xor D_{i+100}
Identify sequences of 0s in E that are longer than 100

- Three threads in the NMP: Generator, Splitter, Counter
- Bmm to split the {Ai}



PartRadio (Vector + stream)

- Input: a float-point stream. Output is a float-point stream
- 3 threads (like ConvEnc)
  - Low pass filter
  - Demodulator
  - Equalizer
- Thread processes 16 elements in parallel (vector)

Data Movement