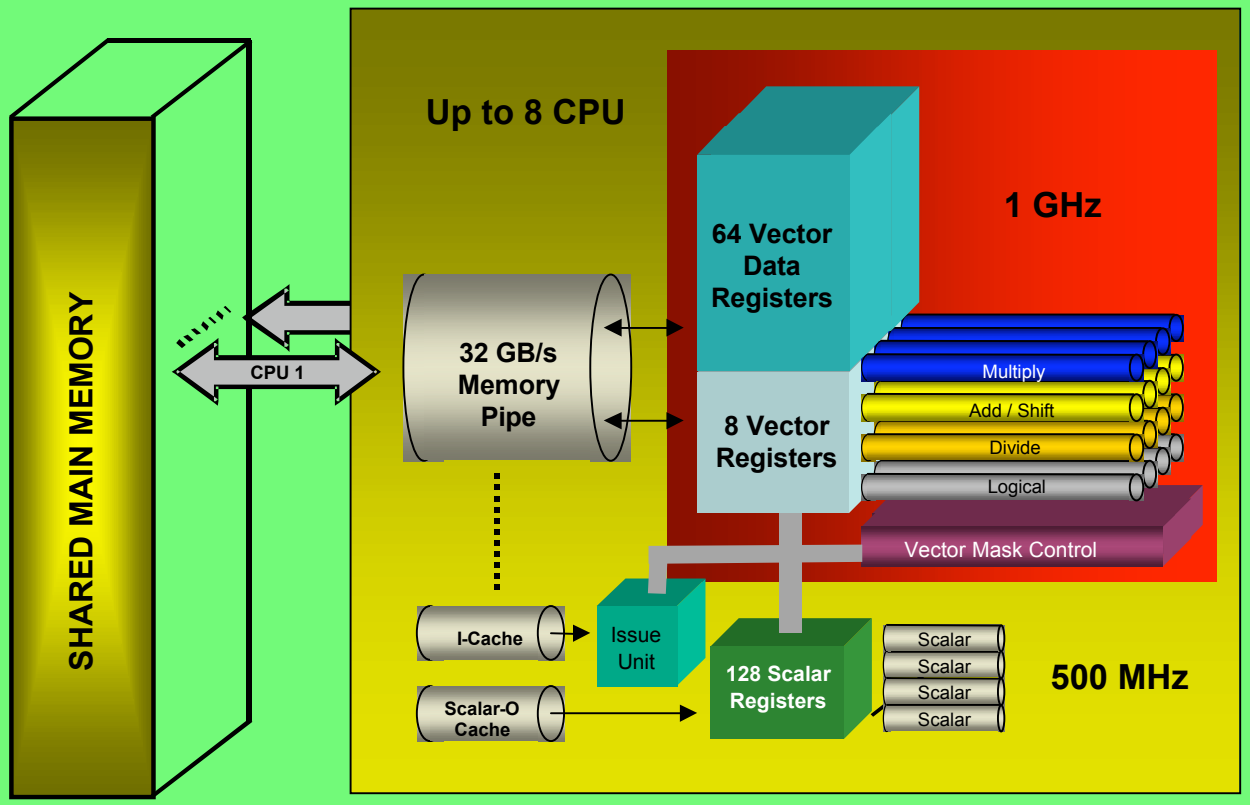
Vectorization of CCM3 and CAM 2.0 on NEC SX systems

David Parks

February 6, 2003

NEC Corporation

SX-6 Processor



SX-6 Processor con't

- Scalar
  - 1 Gflop/s performance
  - 4 way superscalar issue
  - 2 integer units
  - 2 floating point units
  - 128 64-bit general purpose registers
  - 64KB I-cache
  - 64KB O-cache

SX-6 Processor con't

- Vector
  - 8 Gflop/s performance
  - 4 Vector pipeset
  - 8 Vector arithmetic registers
  - 64 Vector data registers
  - 1 Floating add
  - 1 Floating multiply
  - 1 Floating divide (full)
  - Preissue of memory loads
  - 32 GB/S memory interface (4x64 bit words / clock)

CCM/CAM on Vector Systems

- Compile
  - Port CCM/CAM to host architecture
- Profile
- Vectorization
  - Compiler assisted
  - Manual
    - Discussion/examples focused on Fortran 90

Porting

- Usually quite trivial
  - 32 bit integers
  - 64 bit floats
  - Some modification of memory management interface

Profiling Tools

- Flow trace
  - Compiler instrumentation for single and parallel)
  - Statistical sampling of P-register ( )

Vectorization Catagories

- Trivial
  - Vestigial
    - Compiler directives
  - Automatic function inlining
  - Loop exchange
- Moderate
  - Manual inlining
  - Loop fusion
  - Repartitioning day/night time elements vectors

Categories con’t

- Advanced
  - Blocking for vector data registers (SX specific)

Vectorization - Trivial

- Vestigial
  - Find remnants of previous vector code
    - Typically change to
    - Example (CCM3.6.6 – GRCALC.F):

Gather/scatter Lists

- Guarded Operations
  - Example (CCM3 – soih2o.F)
  - Find mask vector of valid points to process

Gather/scatter Lists – con't

- Without no dependency directive

- With no dependency directive

Gather/Scatter con’t

- Depending on truth density, linear operations using mask register may be more efficient.

Vectorization – Trivial con't

- Using instrumentation tools, find high frequency routines (functions) and use compiler's automatic inlining capabilities -
- Example (CAM 2.0_dev14 radae.F90):

radabs

radabs – before & after

Before inlining:

After inlining:

Improvement:

15X

Trivial – con't

- Loop exchange – change order of inner and outer loops.
- Example (CCM3.6.6 linemsbc.F)
  - Original (inner length = 18):
  - Optimized (inner length = 128, 256, …)

What Compilers Can’t Do

Moderate

- Manual inlining (where compiler cannot inline for a variety of reasons).
- Example (CAM2.0_dev14 cldwat.F90):

Manual inlining – con't

- Source to real function estblf in wv_saturation.F90:

- cannot be automatically inlined because and are private to module

Manual inlining – con't

- Make and visible in module
- Change external function reference to inline statement function (cldwat.F90):

Loop Fusion

- Join one or more loops to increase computational intensity (Flops/Memory references).
- Example (ccm3.6.6 radabs.F):

Loop Fusion – con’t

- Radabs - optimized:

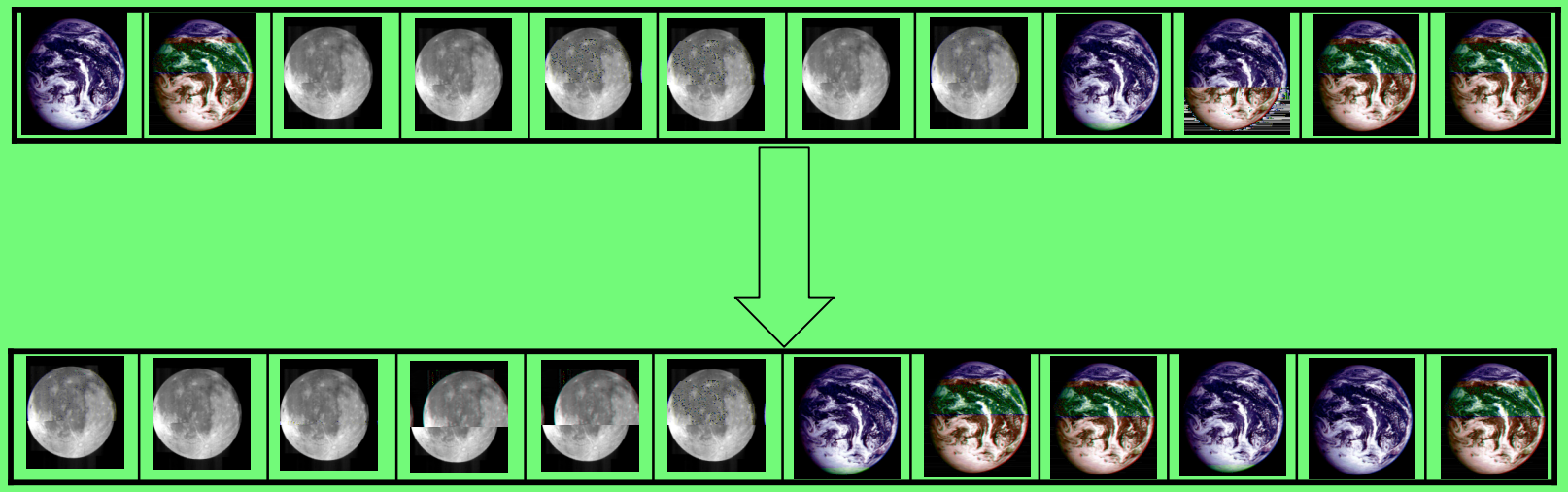
Combine split partitions

- When routines exclude day or night time points and there are 3 partitions (day,night,day), it is sometimes beneficial to swap elements so that all day points are grouped together

- Example (CCM3.10.16 radcswmx.F90)

Combine split partitions

- Swap day/night elements



Combine split partitions

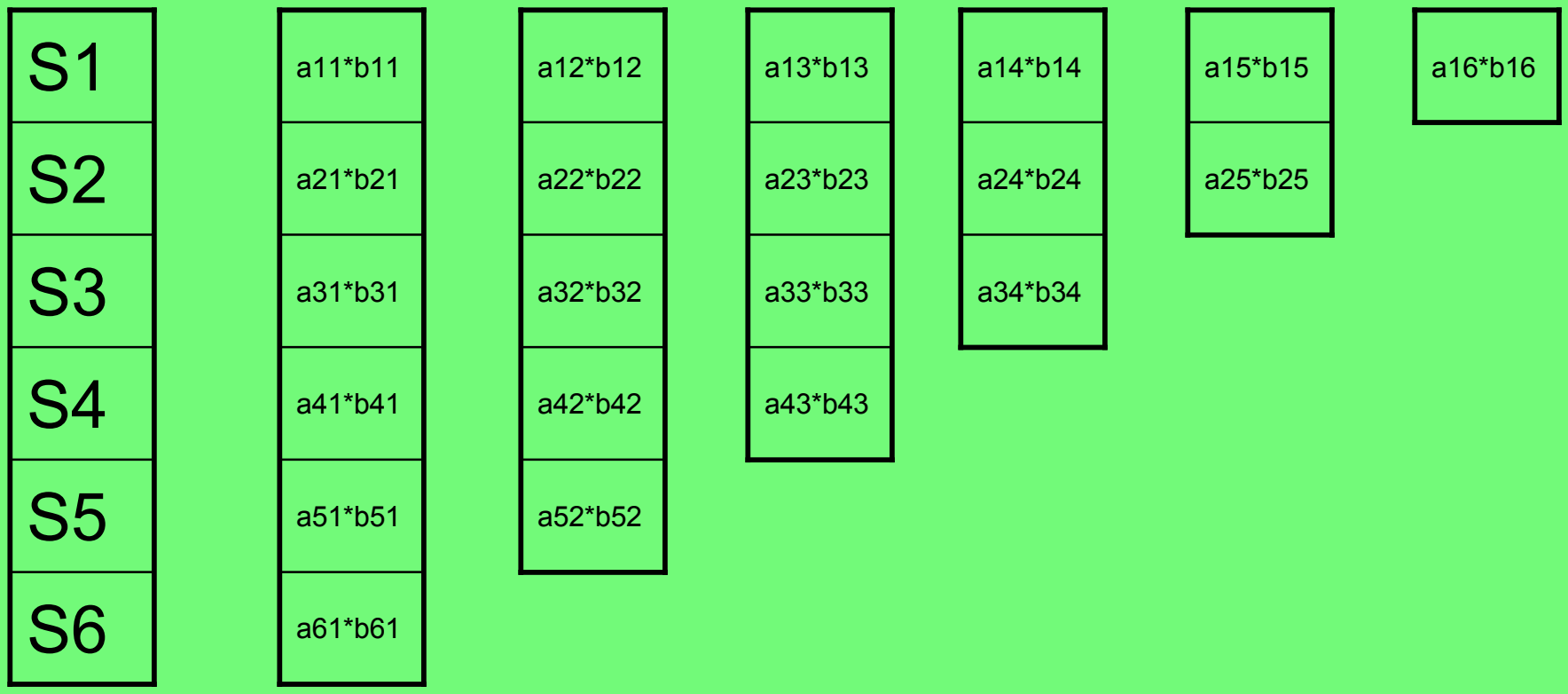
Using Vector Data Registers

- Manually ‘block’ vectors to map to vector data registers.
- 64 vector data registers, 256x64bit each
- No memory traffic to/from vector data registers
- User assignable using

  array_name

- Example (CCM3.6.6 grcalc.F)

General Idea



Vector Data Registers – con't

Vector Data Registers – con't

Vector Data Registers – con't

- Using CCM3.6.6 Eulerian dynamical core
- Grcalc original innermost loop
  - 12 output sums (12 loads + 12 stores)
  - 8 input arrays (8 loads)
  - 27 Floating point operations
  - Computational intensity: 27/32 = 0.84
- Using VDRs
  - 8 input
  - 27 Floating point operations
  - Computational intensity: 27/8 = 3.37

Summary

- Go searching for vestigial vector code
- Use compiler/runtime tools to identify poorly vectorized regions (subroutines, functions, loops)
- Whenever possible use compiler capabilities to vectorize scalar loops (inlining, compiler directives, …)

Summary – con't

- Consider using mask register instead of gather/scatter operations
- Try using vector data registers to block routines with low computational intensity
- Reduce, reduce, reduce, memory traffic
- Increase, increase, increase, computational intensity

END