Idaho National Engineering and Environmental Laboratory

# **Extreme Vectorization in RELAP5-3D**

Dr. George L. Mesina and Peter P. Cebull

May 20, 2004



### **Vectorization Task Overview**

- Background
- Task description
- Analysis & vectorization inhibitors
- Optimization methods
- Extreme vectorization speed-up results
- Summary



### Background

- A recent trend in HPC is a return to vector computing
- INEEL is optimizing its codes for use on vector machines
  - INEEL acquired Cray SV1 computers.
  - Improvement of RELAP5-3D vector performance is part of effort.



# Background: RELAP5-3D

- Nuclear power plant safety analysis
- Physics and models includes
  - Multi-dimensional, multi-phase flow
  - Multi-dimensional heat transfer
  - Multi-dimensional neutron kinetics
  - Trips and control systems
  - Component models such as turbines, valves



# Background: RELAP5-3D

- Other application areas
  - Nuclear plant operator training simulators
  - Fusion safety analysis
  - Steam distribution systems
  - Paper and pulp simulators
- Under development since 1970s
  - Fortran coding from every computing era
  - Has reputation as somewhat of a compiler buster



### Analysis

- Apply Cray performance measures to RELAP5-3D.
- 3 subroutines use approximately 1/3 of CPU time for small and normal-sized runs:
  - PHANTJ, PHANTV, FORCES.
  - Small generic PWR model called TYPPWR
  - Normal-sized model of ROSA facility
- In other runs, these 3 subroutines were always in the top 5 for CPU time.



### **Task Description**

- Improve code performance.
  - Potential 50% speed-up via optimization for some problems.
    - Apply to PHANTJ, PHANTV, FORCES
  - Side benefit: increase run speed on scalar machine.



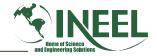
### Analysis

- PHANTV & PHANTJ: have huge loops that do not vectorize:
  - DO-11, DO-111, DO-10
- These had multiple vector inhibitors:
  - Subroutine calls, module use
  - Variable length <u>inner</u> loops
  - Backward GO TO
  - Actual & false recurrence
  - If-tests too deeply nested
  - Loop too long



# **Optimization:** Subprograms

- Inline called subprograms.
  - List them on compiler inline flag.
    - Sufficient for "small" subprograms.
  - For "large" subprograms, must also use a source code pre-compiler directive.
    - Add directive at top of calling subprogram listing the subprograms to inline.



### **Optimization:** Subprograms

- Subprogram inlining often introduces new vector inhibitors to the loop.
  - From previous inhibitors list, active I/O stmts.
  - Mismatched data types in call argument list.
    - Allowed except for interfaces & inlining
    - Caused by RELAP5 FA-array equivalence
    - Rewrite declarations or copy data before and after call



# **Optimization: Modules**

- Proper use of modules.
  - Compile the module with a flag that says it is allowed to be inlined.
  - Do not use allocatable arrays in inlinable modules.
    - Move inlinable subprograms to separate file.
    - Or Mover allocatable array to another module.



### **Optimization Problem: Extreme Loop Length**

- Inlining: effective loop length is huge.
  - 4900 executable lines for PHANTV "DO 11".
  - 7100 executable lines for PHANTJ "DO 10".
  - These are executed hundreds to a few thousand times.
  - RELAP5 has long loops with few iterations.
- Typical vector loops are shorter with many iterations.
- Compiler seems to run out of internal storage for analysis of huge loops.

# Extreme Vectorization of Long Loops

- Technique
  - Use "aggressive" compiler flag with compiler directive CONCURRENT.
- Compiler timings (Cray SV1) with the compiler flags and directives.
  - PHANTV uses 500 seconds.
  - PHANTJ uses 700 seconds.



# **Optimization: Inner Loops**

- A loop with inner loops can vectorize.
  - All inner loops must vectorize & have fixed length.
- The huge loops in PHANTV & PHANTJ have variable length inner loops:
  - Huge loop over control volumes, inner loop over junctions of the volume.
  - Search a general table prior to interpolation.



# **Optimization: Inner Loops**

- Technique 1: Perform variable-length calculations before loop (if possible).
  - Store results in temporary arrays.
  - Access temporary arrays in the huge loop.
- Technique 2: Convert search to direct index calculation
  - Replace non-uniform mesh with uniform mesh.
    - Include all non-uniform mesh points in uniform mesh.
    - Generate new table via interpolation.
  - Calculate subinterval directly; careful at endpoints!



#### **Optimization: Recurrence Elimination**

- Recurrence: when data in one iteration depends on data from another iteration of the same loop.
- Recurrence 1: PHANTJ FIDXUP calculation
  - Inner loop sums series of complicated calculations.
  - One scalar result, FIDXUP, for each junction.
- Technique: Move recurrence loop to before huge loop (if possible).
  - Promote FIDXUP to array.
  - Move FIDXUP calculation loop to a junction loop before huge loop.



### **Optimization:** Recurrence

- Technique 2: Replace bisection with spline evaluation of inverse function.
  - Find Burrington angle calculation: b sin(b) =  $2\pi\alpha_g$ 
    - Bisect to get zero of:  $f(b; \alpha_g) = 2\pi\alpha_g b + sin(b)$
  - Calculate cubic spline during input processing.
    - Via bisection get f<sup>-1</sup>(b;  $\alpha_q$ ) at 200 pts.
    - Generate cubic spline via DeBoor B-splines.
  - For any  $\alpha_{q}$ , directly calculate subinterval.
    - Evaluate cubic spline there to find angle b.



# **Optimization: Backward GoTo**

- Backward GoTo's prevent vectorization.
- PHANTV & PHANTJ have backward GoTo 1521
  - Both have complex if-tests at entry and exit.
  - Body of the if-test executed no more than twice.
    - First pass horizontal or vertical coefficients.
    - Second the other coefficients for smooth interpolation if flow angle in transition region.

– If-test body 660 lines in PHANTJ, 1220 in PHANTV

Cannot duplicate body - maintenance issue



# **Optimization: Backward GoTo**

- Do while loop with exit rejected for variable length.
- Do loop of length 2 with exit rejected by compiler.
- Technique: Place coding encapsulated by Backward Go To in a subroutine.
  - Over 100 variables to pass to subroutine.

Original PHANTV	Modified PHANTV			
1521 if (Test 1) then Block of Code	1521 call stratv if ( <b>Test 2</b> ) call stratv			
endif if ( <b>Test 2</b> ) go to 1521	subroutine stratv if ( <b>Test 1</b> ) then Block of Code endif			

### **Optimization of 2 Kinds of False Recurrence**

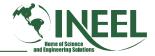
- 1. All RELAP5 arrays equivalence to FA array
  - Appears to be recursive to compiler.
  - Apply compiler directive IVDEP or CONCURRENT.
- 2. Statements with small increments seem recursive to the compiler
  - EG: fwfxaf(ix+1) = fwfxaf(ix); costhe(ix+2) = costhe(ix).
  - Not recursive: ix increments by 279.
  - Solution: move such statements outside the huge loop to a separate loop (if possible).



### **Optimization: Deep If-Tests**

- Compiler handles only so many levels of nesting.
   Else and elseif branches count as levels too.
- Remove nesting levels via logical variables.
- Eliminated up to 6 levels as needed.

```
Nested if-test
                                 No Nesting
                                 LV0 = void>0
if (void>0) then
                                 LV1 = LV0 .and. void<1
  if (void<1) then
                                 LVCN = LV1 .and. btest(c,n)
    if ( btest(c,n) ) then
                                 LVCNH = LVCN .and. hmap
      CALCULATIONS 1
                                 if (LVCN) then
      if (hmap) then
                                   CALCULATIONS 1
        CALCULATIONS 1.1
                                 if (LVCNH) then
    else
                                   CALCULATIONS 1.1
      CALCULATIONS 2
                                 if (LV1 .and. .not.LVCN) then
                                   CALCULATIONS 2
```



### Speed-up Objective Met

Test Case	Phan	tv MFL	OPS	Phantj MFLOPS		
	Orig.	Vector	V/O	Orig.	Vector	V/O
TYPPWR	13.5	57	4.2	10.9	66.5	6.1
ROSA	18.8	76.5	4.1	10.6	88.6	8.4
AP600	14.9	92.5	6.2	10.3	127.6	12.4
3Dflow15	16.0	98.5	6.2	9.7	136.1	14.0

- Speed is measured in MFLOPS
- Best Performance in RED.
- RELAP5-3D speedup of 51% achieved.
  - Inlining pulled non-vector code into vector loops.
  - Increased % of code in PHANTV & PHANTJ.



### Summary

- Task was to vectorize RELAP5-3D to improve run speed.
- Many techniques for vectorizing extremely long loops were presented.
- Run speed increased up to 51%.
- Legacy Fortran codes should be re-examined for vector speedup.