

Optimization of the Salvo code on the Origin 2000

Bart van Bloemen Waanders* & Jan Thorbecke

Salvo consortia meeting

June 26, 1998

Part 1: Architecture of the Origin 2000

Part 2: Performance tools on the Origin 2000

Part 3: Performance optimization for salvo

Part 4: Shared Memory version of salvo

Performance tools on the Origin 2000

- **perfex** for an overall view on computational costs
- **speedshop** for detailed analysis on source code level
- **-S compiler option** for software pipelining

perfex: description

perfex prints the values of 32 hardware performance counters of the R10000.

- 21 = Graduated floating point instructions
- 25 = Primary data cache misses
- 26 = Secondary data cache misses

Options:

- -a Multiplex over all events, projecting totals.
- -y Report statistics and ranges of estimated times per event.
- -x Count at exception level (as well as the default user level).

```
SGI> perfex -x -y -a a.out
```

perfex: example

```
SGI> mpirun -np 1 perfex -x -y -a -o output salvo < shot.prm
```

Event Counter Name	Counter Value	Typical Time (sec)
21 Graduated floating point instructions.....	28920012288	148.307755
25 Primary data cache misses.....	1430674576	66.104502
26 Secondary data cache misses.....	37043312	14.342411

Statistics

```
=====
```

L1 Cache Line Reuse.....	14.362031
L2 Cache Line Reuse.....	37.621670
L1 Data Cache Hit Rate.....	0.934904
L2 Data Cache Hit Rate.....	0.974108
Time accessing memory/Total time.....	0.673777

Note that for salvo there are no 'strange' things reported by perfex.

speedshop: description

Integrated package of performance tools to run performance experiments on executables, and to examine the results of those experiments.

- `ssrun` - set up and run a process to collect SpeedShop performance data
 - `-[f]pcsamprc` emphasizes functions that cause cache misses
 - `-[f]gfp_hwc` *statistical* PC sampling, of the graduated floating-point instruction counter
 - `-ideal` uses basic-block counting, by instrumenting the executable
- `prof` - analyze SpeedShop performance data

```
SGI> ssrun -fgfp_hwc a.out
```

```
SGI> prof -h -b .ideal
```

speedshop: example

```
SGI> mpirun -np 1 sssrun -fgfp_hwc salvo < shot.prm
```

```
-----  
Function list, in descending order by counts  
-----
```

[index]	counts	%	cum.%	samples	function (dso: file, line)
[1]	11446256160	39.8%	39.8%	1746720	pipe_single (salvo: pipe_single.F, 99)
[2]	5106851195	17.8%	57.6%	779315	PASSF4 (salvo: fftpack.f, 1221)
[3]	5087585375	17.7%	75.2%	776375	PASSB4 (salvo: fftpack.f, 909)
[4]	2332999060	8.1%	83.4%	356020	phase_correction (salvo: filter_li.F, 259)
[5]	1754526432	6.1%	89.5%	267744	PASSB5 (salvo: fftpack.f, 961)
[6]	1747560593	6.1%	95.5%	266681	PASSF5 (salvo: fftpack.f, 1273)
[7]	723313587	2.5%	98.0%	110379	thin_lens (salvo: thin_lens.F, 36)
[8]	306202031	1.1%	99.1%	46727	__cosf (libm.so: fcos.c, 93)
[9]	155050533	0.5%	99.6%	23661	image (salvo: image.F, 34)
[10]	36133242	0.1%	99.8%	5514	__vcosf (libm.so: vfcos.c, 77)
[11]	30654934	0.1%	99.9%	4678	__vsinf (libm.so: vfsin.c, 77)
[12]	9678781	0.0%	99.9%	1477	RADF5 (salvo: fftpack.f, 225)
[13]	9174200	0.0%	99.9%	1400	RADF3 (salvo: fftpack.f, 135)
[14]	4442934	0.0%	100.0%	678	vel_interp (salvo: vel_interp.c, 47)
[15]	4148049	0.0%	100.0%	633	RADF4 (salvo: fftpack.f, 171)
[16]	3158546	0.0%	100.0%	482	CFFTI1 (salvo: fftpack.f, 664)
[17]	1690674	0.0%	100.0%	258	apply_encoding (salvo: apply_encoding.F, 31)
[18]	878102	0.0%	100.0%	134	scatterFrequencies (salvo: scatterFreq.c, 74)
[19]	203143	0.0%	100.0%	31	__pow (libm.so: pow.c, 198)
[20]	183484	0.0%	100.0%	28	filter_li (salvo: filter_li.F, 50)
[21]	150719	0.0%	100.0%	23	RFFTI1 (salvo: fftpack.f, 36)
[22]	72083	0.0%	100.0%	11	hdr_get (salvo: segyhdr.c, 80)
[23]	32765	0.0%	100.0%	5	segy_getxyz (salvo: segyio.c, 545)
[24]	32765	0.0%	100.0%	5	scatterData (salvo: scatterData.c, 62)
[25]	19659	0.0%	100.0%	3	fftData (salvo: fftData.c, 151)
[26]	13106	0.0%	100.0%	2	__libm_dcis (libm.so: dcis.c, 179)
[27]	6553	0.0%	100.0%	1	memcpy (libc.so.1: bcopy.s, 329)
[28]	6553	0.0%	100.0%	1	timeline_ (salvo: timeline.c, 73)
[29]	6553	0.0%	100.0%	1	seis_fftlens (salvo: seiff.c, 87)
[30]	6553	0.0%	100.0%	1	initvars (salvo: initvars.F, 76)

compiler -S option: description

Compile the specified source programs and leave the symbolic assembly language output in corresponding files suffixed with .s.

Usefull to check how effectively the processor's hardware resources are being used in the schedule generated by the compiler. Specially software pipelining and loop unrolling are reported.

```
SGI> f77 -O3 -S file.f
```


compiler -S option: example

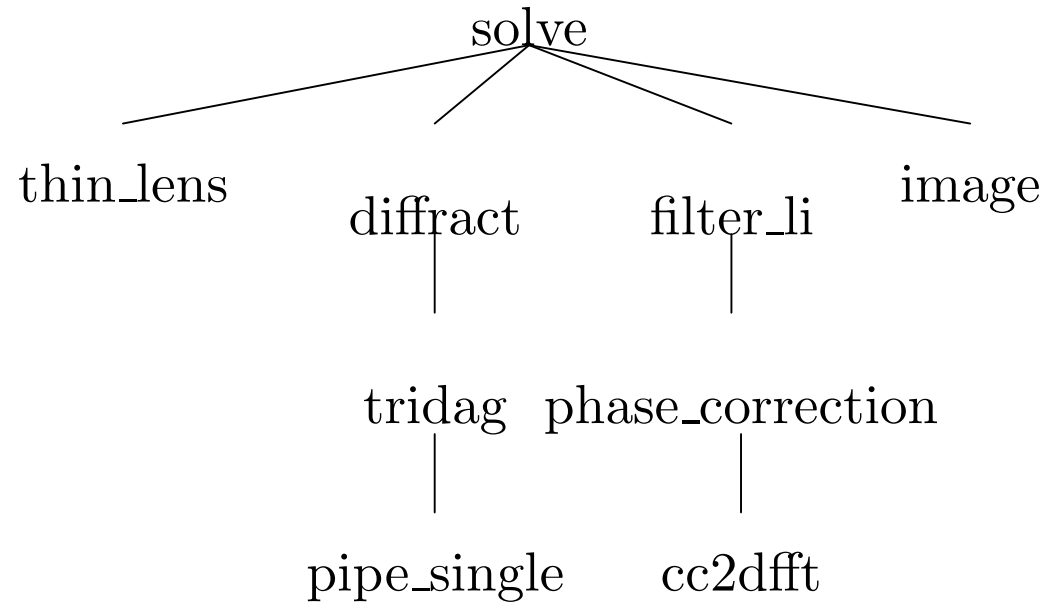
```
SGI> f77 -O3 -S pipe_single.f
```

```
#<swps> Pipelined loop line 678 steady state
#<swps>
#<swps> 100 estimated iterations before pipelining
#<swps> Not unrolled before pipelining
#<swps> 8 cycles per iteration
#<swps> 8 flops ( 50% of peak) (madds count as 2)
#<swps> 6 flops ( 37% of peak) (madds count as 1)
#<swps> 2 madds ( 25% of peak)
#<swps> 6 mem refs ( 75% of peak)
#<swps> 3 integer ops ( 18% of peak)
#<swps> 15 instructions ( 46% of peak)
#<swps> 1 short trip threshold
#<swps> 4 integer registers used.
#<swps> 6 float registers used.
#<swps>
#<swps> 6 min cycles required for resources
#<swps> 8 cycles in minimum schedule found
.....
#<swps> Pipelined loop line 387 steady state
#<swps>
#<swps> 50 estimated iterations before pipelining
#<swps> 2 unrollings before pipelining
#<swps> 8 cycles per 2 iterations
#<swps> 8 mem refs (100% of peak)
#<swps> 4 integer ops ( 25% of peak)
#<swps> 12 instructions ( 37% of peak)
#<swps> 2 short trip threshold
#<swps> 9 integer registers used.
#<swps> 8 float registers used.
```

Note that loops which include a function call are not pipelined.

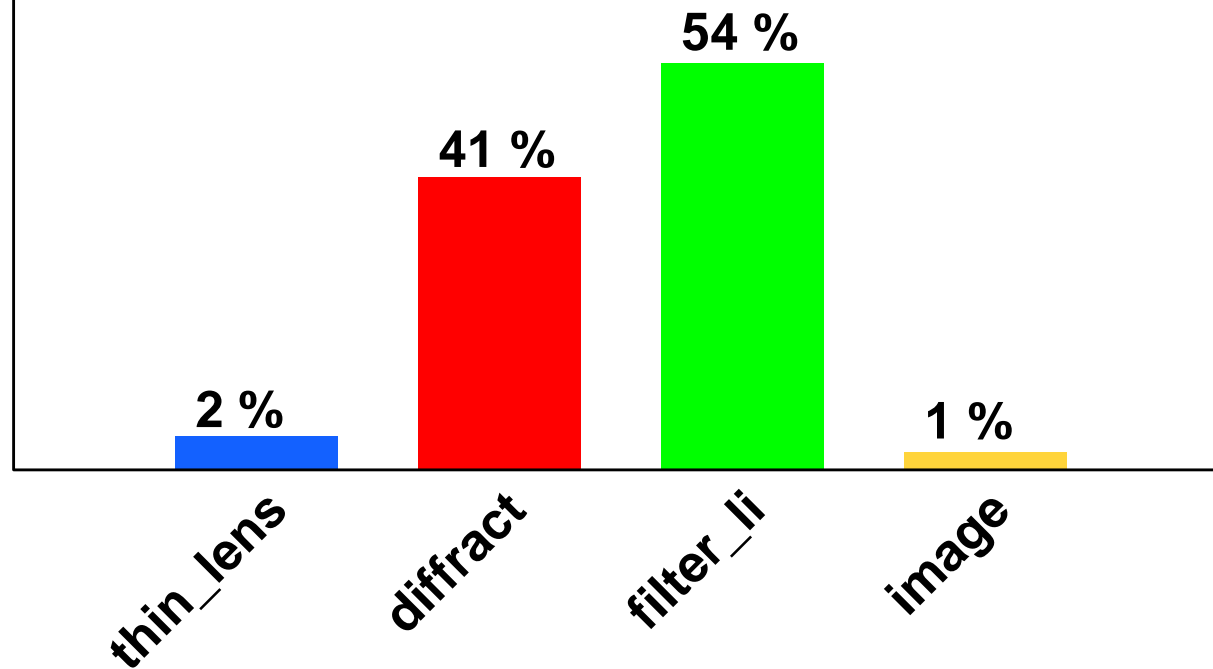
Performance optimization for salvo

Computational scheme:



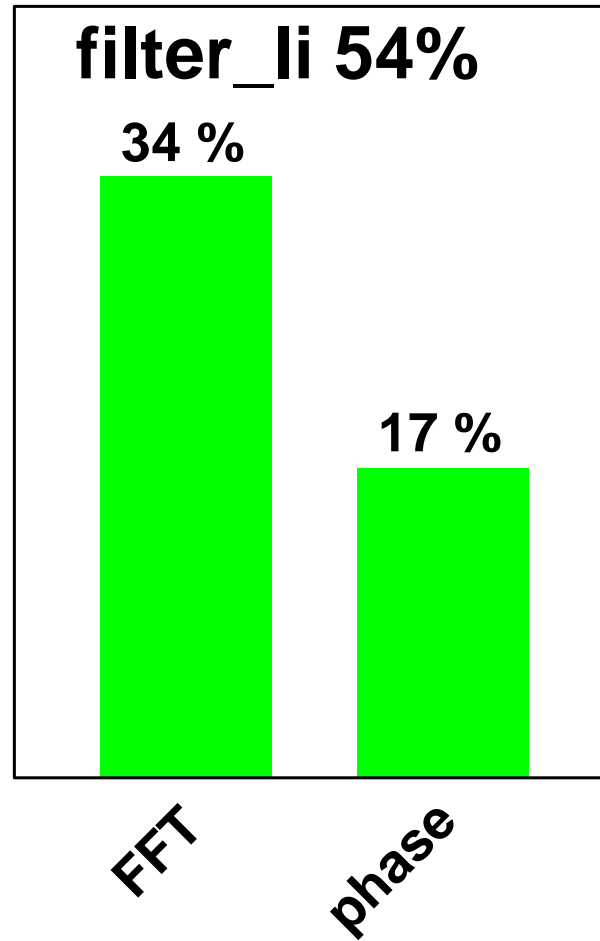
Performance optimization for salvo

solve 98%



profiled with *ssrun -ideal* experiment

Performance optimization for salvo



Performance optimization for salvo

- use SGI's **SCS** library (-lscs_mp) to do the FFT's
- rearrange calculations in pipe_single.f
- replace complex arrays with real arrays (*to be done*)
- loop level optimization (*to be done*)

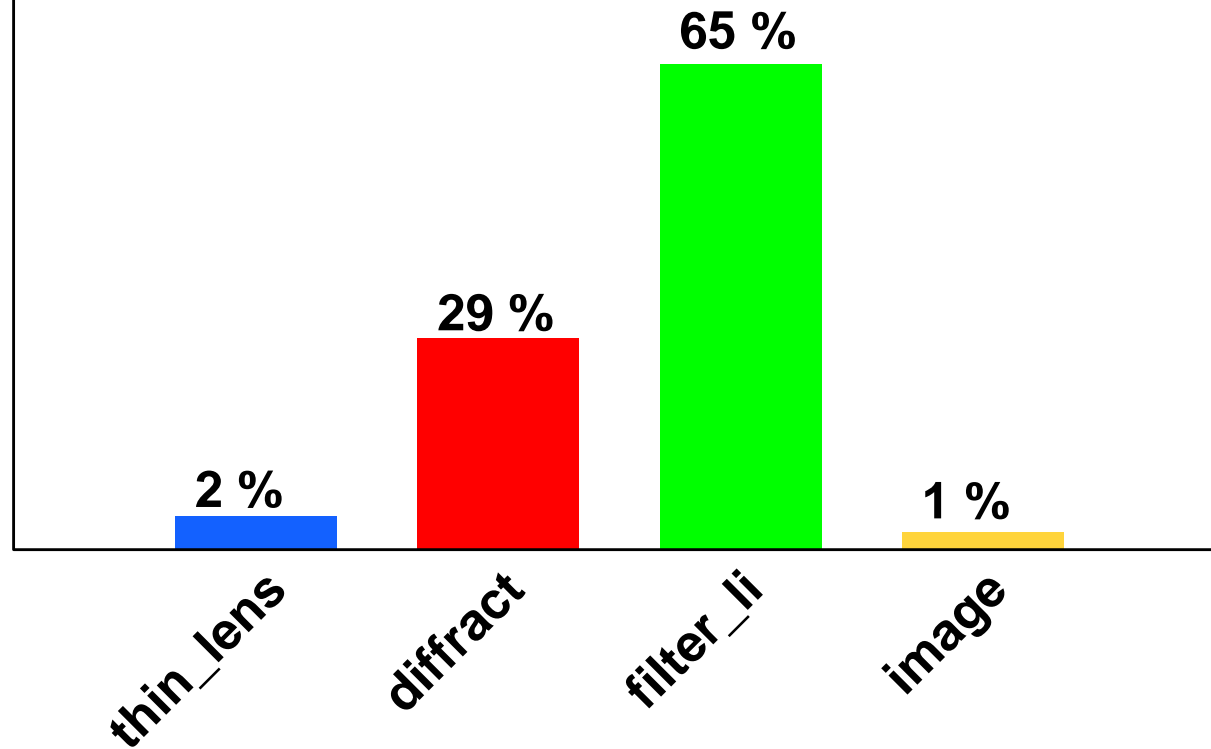
Performance optimization for salvo

	Time (s)	instructions	Gflop*	Gintop	Mflop/s
Original	206	69.10 ⁹	37	11	143
replaced FFT's	137	67.10 ⁹	37	10	146
rearrange calc's	116	62.10 ⁹	35	10	172
shared memory	113	59.10 ⁹	35	9	180

The times given are for one CPU; computing one depth step of the migration of one shot record of the Sandia_OBC data-set in the frequency range [7.5, 90.0] Hz. One depth slice is 295x295 (nx~~x~~ny) points and 297 frequencies are calculated. * Note that one floating point cycle may consist of one add and a mult.

Performance optimization for salvo

optimized solve 97%



profiled with *ssrun -ideal* experiment

Performance optimization for salvo

Scaling of frequency parallelism (MPI partially code-optimized)

# CPU's	Time (s)
1	116
2	58.9
4	30.8
8	15.6
16	8.2
32	4.7
64	-

Note that these results were obtained by running on a non-dedicated machine.

Shared memory version of salvo

Migration algorithm:

```
for (iz=0; iz<nz; iz++) {  
    for (ifreq=0; ifreq<nw; ifreq++) {  
        copy freq slices data[ifreq][ny][nx] to local array[ny][nx]  
        extrapolate source and receiver fields (call salvo)  
        imaging condition (call image)  
        store extrapolated freq slice back in data[ifreq][ny][nx]  
    }  
    add all images from all frequencies  
    write total image for iz to disk  
}
```

Note that the depth loop can be moved inside the frequency loop without rewriting the code.

Single CPU performance

Method	total time [s]		time/depth_step [s/m]	
	195 Hz	250 Hz	195 Hz	250 Hz
Original	1867	-	203	-
Optimized	1066	-	114	-
Shared Memory	969	743	107	85

For the shared memory version a smaller FFT length 320 =_i 300 is used.

Using the Sandia_OBC data set and the following parameters:

Output image: $n_x * n_y * n_z = 295 * 295 * 10$ (size of the output image) $dy=82$
 $dx=82$ $dz=25$ $filter_type=1$ $image_type=0$ (=Correlation)

velocity input file: $n_{xv}=102$ $x_{vmin}=24272$ $dxv=656$ $n_{yv}=75$ $y_{vmin}=76096$
 $dyv=656$ $n_{zv}=97$ $z_{vmin}=0$ $dzv=150$

frequency range: $f_{min}=7.5$ $f_{max}=90$ ($nf=297$)

compiler: 7.2.1 with -O3 -mips4 -n32 OS: IRIX 6.5 CPU: MIPS r10000 at
195 MHz and 4 MB secondary cache

Scaling of frequency parallelism

# CPU's	<i>SMP</i> ¹	<i>SMP</i> ²	MPI	MPI(original)
1	969		1066	1867
2	511		544	962
4	268		284	505
8	134		141	257
16	71		77	128
32	44		44	74
64	-	-	-	-
128	-	-	-	-

Total wallclock time (user+system) is shown. 1) 195 Mhz R10k with 4 MB of secondary cache 2) 250 Mhz R10k with 4 MB of secondary cache 3) 300 Mhz R12k with 8 MB of secondary cache

Concluding remarks

-
-
-