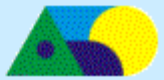
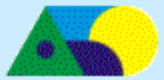
Debugging and profiling, how and why

Jure Jerman, Environmental Agency of Slovenia



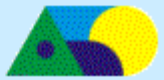
Outline

- Introduction
- Debugging
- Profiling
- Conclusions

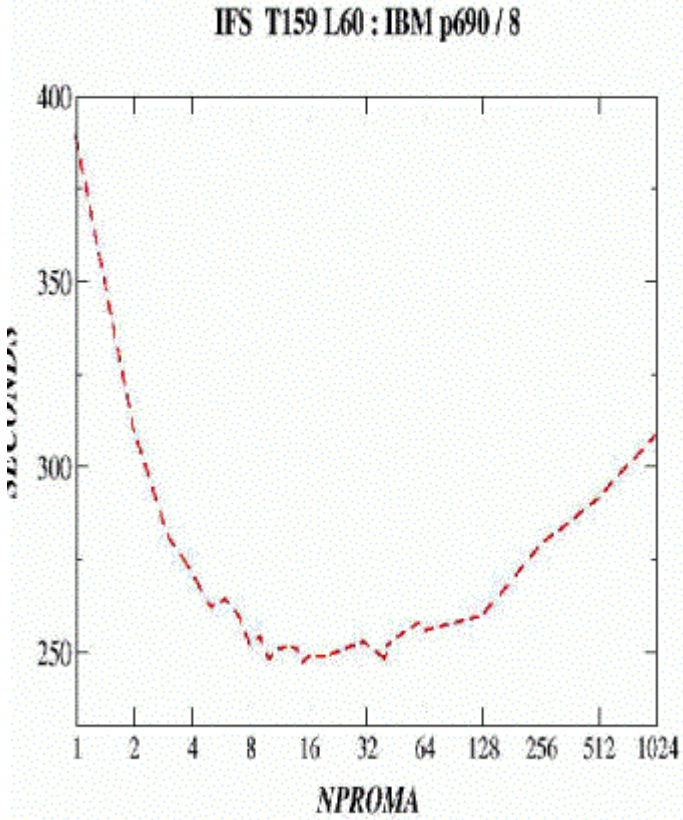


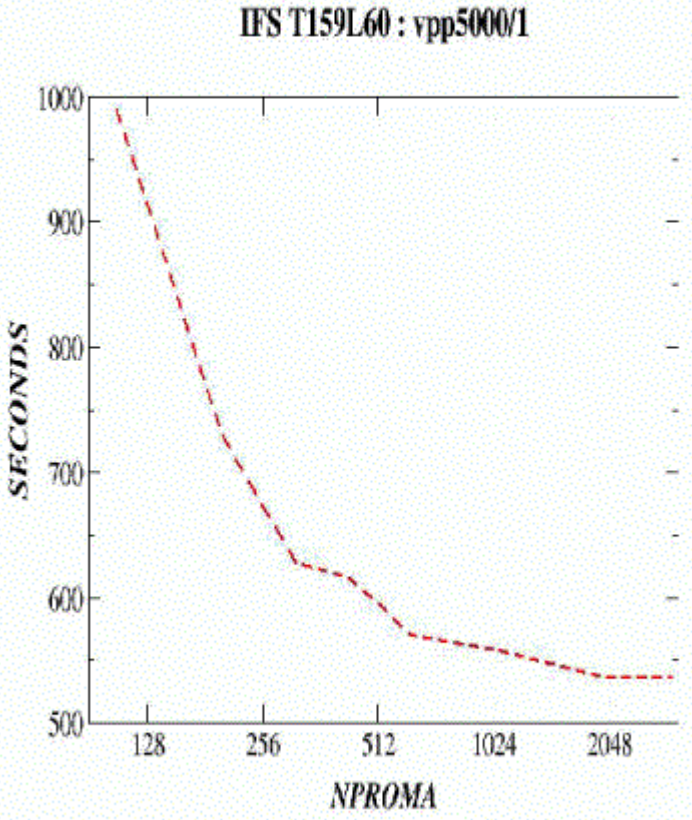
Introduction

- ALADIN is very complex piece of code (together with Arpege/IFS) ~500.000 lines of code, 3000-4000 routines
- Succesfuly tested on many hardware platforms with many compilers (from PC workstations to Supercomputers)
- However, every phasing with new IFS/Arpege code, port to new environment or port of the new cycle, might be a night mare for code maintainers.
- This work can be easier with the usage of right tools
- Debugging tools to make the process easier
- Profiling tools, to validate if the work has been done right from cycle to cycle



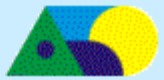
NPROMA


IFS T159 L60 : IBM p690 / 8
400
350
300
250
1
2
4
8
16
32
64
128
256
512
1024
NPROMA


IFS T159L60 : vpp5000/1
1000
900
800
700
600
500
SECONDS
128
256
512
1024
2048
NPROMA

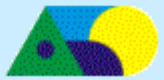
Debugging

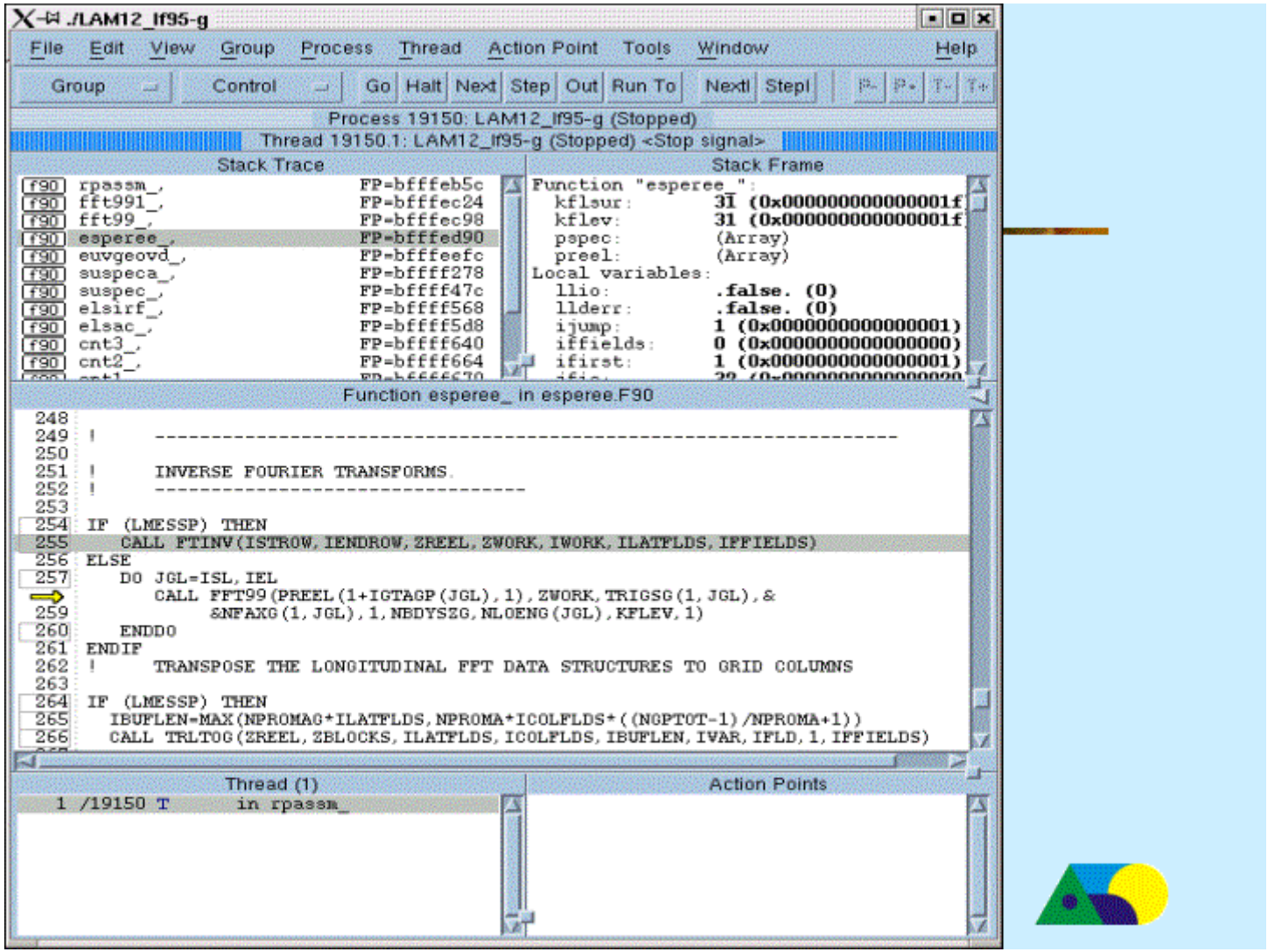
- Procedure is specific for every hw/sw environment
- Mostly used method is printing out the values from the code
  - Time consuming
  - Not to elegant
- Good tools for code navigation are very useful (like fortran2html tool developed by Eric Savault)

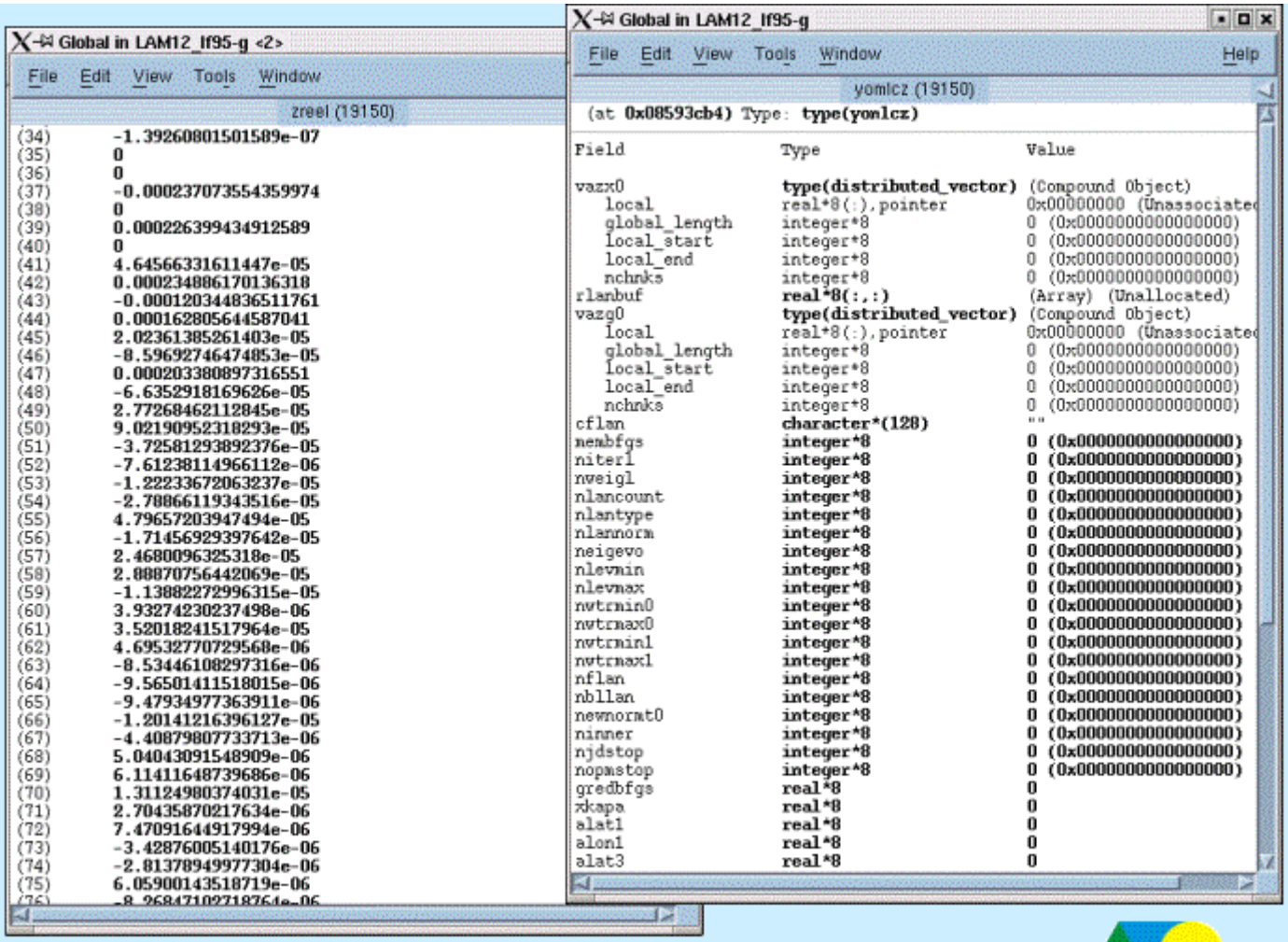


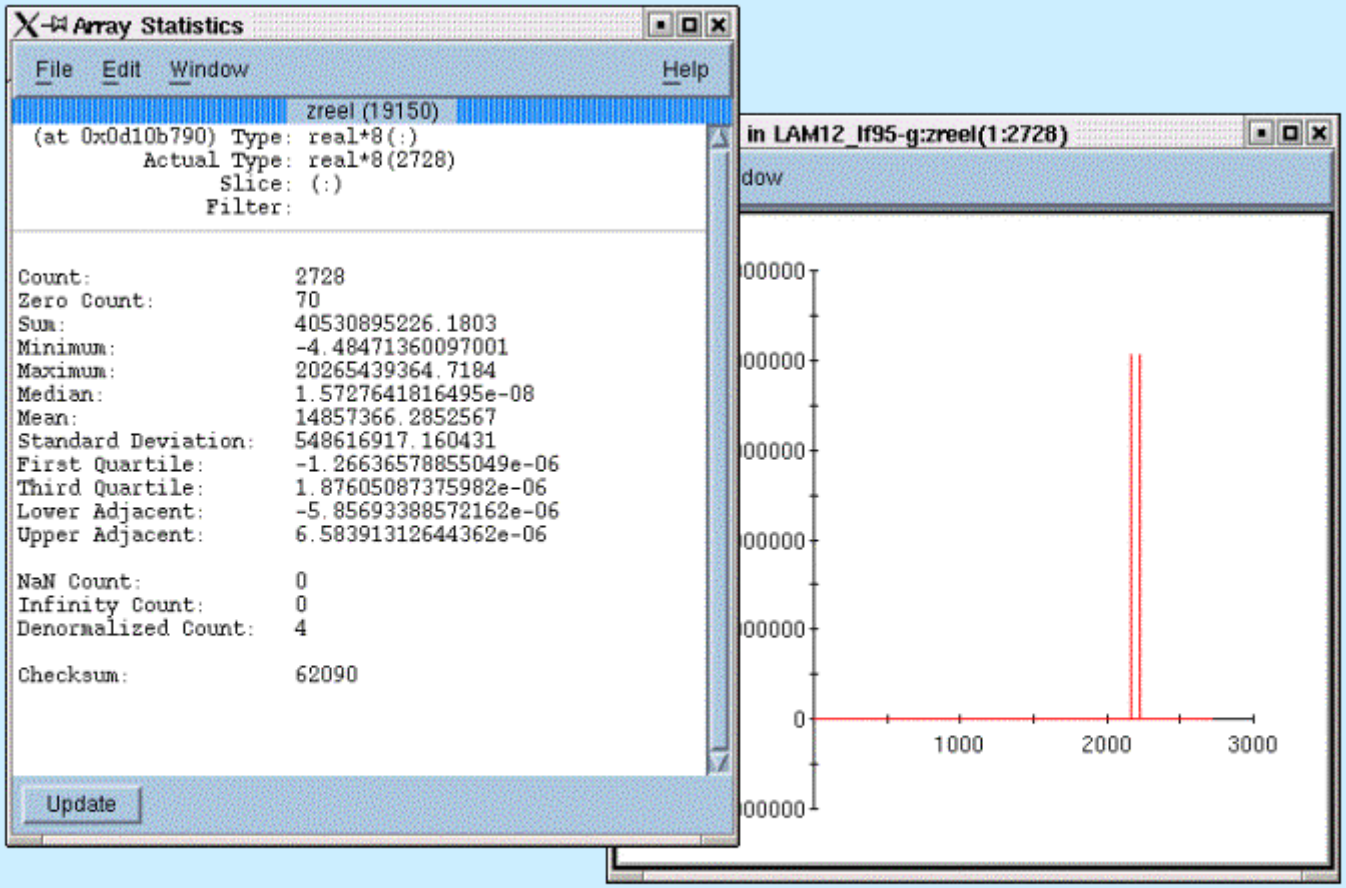
Debugging with TotalView

- Commercial debugger available for most of hw platforms where ALADIN is running, including Linux
- Not very cheap, but many times worth of its money
- Graphical Users Interface (GUI) and Command Language Interface (CLI) for automated debugging
- Capable of debugging OpenMP (multithreaded), MPI applications



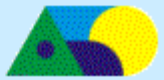

./LAM12_lf95-g
File Edit View Group Process Thread Action Point Tools Window Help
Group Control Go Halt Next Step Out Run To Nextl Stepl
Process 19150: LAM12_lf95-g (Stopped)
Thread 19150.1: LAM12_lf95-g (Stopped) <Stop signal>
Stack Trace
f90 rpassm_, FP=bfffeb5c
f90 fft991_, FP=bfffec24
f90 fft99_, FP=bfffec98
f90 esperee_, FP=bfffed90
f90 euvgeovd_, FP=bfffeefc
f90 suspeca_, FP=bffff278
f90 suspec_, FP=bffff47c
f90 elsirf_, FP=bffff568
f90 elsac_, FP=bffff5d8
f90 cnt3_, FP=bffff640
f90 cnt2_, FP=bffff664
Stack Frame
Function "esperee_":
kflsur: 31 (0x000000000000001f)
kflev: 31 (0x000000000000001f)
pspec: (Array)
preel: (Array)
Local variables:
llio: .false. (0)
llderr: .false. (0)
ijump: 1 (0x0000000000000001)
iffields: 0 (0x0000000000000000)
ifirst: 1 (0x0000000000000001)
Function esperee_ in esperee.F90
248
249 ! ----------
250
251 ! INVERSE FOURIER TRANSFORMS.
252 ! ----------
253
254 IF (LMESSP) THEN
255 CALL FTINV(ISTROW,IENDROW,ZREEL,ZWORK,IWORK,ILATFLDS,IFFIELDS)
256 ELSE
257 DO JGL=ISL,IEL
CALL FFT99(PREEL(1+IGTAGP(JGL),1),ZWORK,TRIGSG(1,JGL),&
259 &NFAXG(1,JGL),1,NBDYSZG,NLOENG(JGL),KFLEV,1)
260 ENDDO
261 ENDIF
262 ! TRANSPOSE THE LONGITUDINAL FFT DATA STRUCTURES TO GRID COLUMNS
263
264 IF (LMESSP) THEN
265 IBUFLEN=MAX(NPROMAG*ILATFLDS,NPROMA*ICOLFLDS*((NGPTOT-1)/NPROMA+1))
266 CALL TRLTOG(ZREEL,ZBLOCKS,ILATFLDS,ICOLFLDS,IBUFLEN,IVAR,IFLD,1,IFFIELDS)
Thread (1)
1 /19150 T in rpassm_
Action Points


Global in LAM12_lf95-g <2>
File Edit View Tools Window
zreel (19150)
(34) -1.39260801501589e-07
(35) 0
(36) 0
(37) -0.00023073554359974
(38) 0
(39) 0.000226399434912589
(40) 0
(41) 4.64566331611447e-05
(42) 0.000234886170136318
(43) -0.000120344836511761
(44) 0.000162805644587041
(45) 2.02361385261403e-05
(46) -8.59692746474853e-05
(47) 0.000203380897316551
(48) -6.6352918169626e-05
(49) 2.77268462112845e-05
(50) 9.02190952318293e-05
(51) -3.72581293892376e-05
(52) -7.61238114966112e-06
(53) -1.22233672063237e-05
(54) -2.78866119343516e-05
(55) 4.79657203947494e-05
(56) -1.71456929397642e-05
(57) 2.4680096325318e-05
(58) 2.88870756442069e-05
(59) -1.13882272996315e-05
(60) 3.93274230237498e-06
(61) 3.52018241517964e-05
(62) 4.69532770729568e-06
(63) -8.53446108297316e-06
(64) -9.56501411518015e-06
(65) -9.47934977363911e-06
(66) -1.20141216396127e-05
(67) -4.40879807733713e-06
(68) 5.04043091548909e-06
(69) 6.11411648739686e-06
(70) 1.31124980374031e-05
(71) 2.70435870217634e-06
(72) 7.47091644917994e-06
(73) -3.42876005140176e-06
(74) -2.81378949977304e-06
(75) 6.05900143518719e-06
Global in LAM12_lf95-g
File Edit View Tools Window Help
yomlcz (19150)
(at 0x08593cb4) Type: type(yomlcz)
Field Type Value
vazx0 type(distributed_vector) (Compound Object)
local real*8(:),pointer 0x00000000 (Unassociated
global_length integer*8 0 (0x0000000000000000)
local_start integer*8 0 (0x0000000000000000)
local_end integer*8 0 (0x0000000000000000)
nchnks integer*8 0 (0x0000000000000000)
rlanbuf real*8(:,:) (Array) (Unallocated)
vazg0 type(distributed_vector) (Compound Object)
local real*8(:),pointer 0x00000000 (Unassociated
global_length integer*8 0 (0x0000000000000000)
local_start integer*8 0 (0x0000000000000000)
local_end integer*8 0 (0x0000000000000000)
nchnks integer*8 0 (0x0000000000000000)
cflan character*(128) ""
nembfgs integer*8 0 (0x0000000000000000)
niterl integer*8 0 (0x0000000000000000)
nveigl integer*8 0 (0x0000000000000000)
nlancount integer*8 0 (0x0000000000000000)
nlantype integer*8 0 (0x0000000000000000)
nlannorm integer*8 0 (0x0000000000000000)
neigevo integer*8 0 (0x0000000000000000)
nlevmin integer*8 0 (0x0000000000000000)
nlevmax integer*8 0 (0x0000000000000000)
nvtrmin0 integer*8 0 (0x0000000000000000)
nvtrmax0 integer*8 0 (0x0000000000000000)
nvtrmin1 integer*8 0 (0x0000000000000000)
nvtrmax1 integer*8 0 (0x0000000000000000)
nflan integer*8 0 (0x0000000000000000)
nbllan integer*8 0 (0x0000000000000000)
newnormt0 integer*8 0 (0x0000000000000000)
ninner integer*8 0 (0x0000000000000000)
njdstop integer*8 0 (0x0000000000000000)
nopmstop integer*8 0 (0x0000000000000000)
gredbfgs real*8 0
xkapa real*8 0
alat1 real*8 0
alon1 real*8 0
alat3 real*8 0


Array Statistics
File Edit Window Help
zreel (19150)
(at 0x0d10b790) Type: real*8(:)
Actual Type: real*8(2728)
Slice: (:)
Filter:
Count: 2728
Zero Count: 70
Sum: 40530895226.1803
Minimum: -4.48471360097001
Maximum: 20265439364.7184
Median: 1.5727641816495e-08
Mean: 14857366.2852567
Standard Deviation: 548616917.160431
First Quartile: -1.26636578855049e-06
Third Quartile: 1.87605087375982e-06
Lower Adjacent: -5.85693388572162e-06
Upper Adjacent: 6.58391312644362e-06
NaN Count: 0
Infinity Count: 0
Denormalized Count: 4
Checksum: 62090
Update
in LAM12_lf95-g:zreel(1:2728)
dow
1000
2000
3000

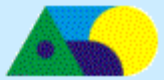
Debugging with standard tools

- Usualy all compilers come with debuggers
- It is not always needed to use very sophisticated commercial debuggers.
- What to do in the case of MPI jobs?
  - The solution is to attach debugger to running process
- Some of mpirun scripts can do this automaticly



Profiling

- Tool to check the time consumption of the code in routine/line
- Good habit to check the timing of code from cycle to cycle
- Some problems could be spoted imidiatly.
- Usage of standard tools like Unix's prof, (gprof) or OS dependent
- Code has to be instrumented (by compilation option) or already compiled

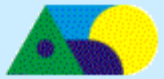


Example of gprof output (1)

Flat profile:

Each sample counts as 0.01 seconds.

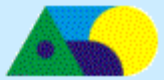
% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
9.46	7.66	7.66	792	9.67	9.67	acraneb_
8.06	14.18	6.52	792	8.23	8.32	accvimp_
6.70	19.60	5.42	14055	0.39	0.39	rpassm_
5.96	24.42	4.82	3168	1.52	1.52	laitri_
3.98	27.64	3.22	1584	2.03	2.03	elascaw_
3.62	30.57	2.93	792	3.70	3.70	accoefk_
3.56	33.45	2.88	792	3.64	35.27	aplpar_
2.84	35.75	2.30	792	2.90	2.90	acdrag_
2.48	37.76	2.01	7467	0.27	1.13	fft991_
2.34	39.65	1.89	5544	0.34	0.34	laitli_
2.16	41.40	1.75	5570	0.31	0.31	sgemmx_
2.14	43.13	1.73	792	2.18	9.34	elarmes_
1.96	44.72	1.59	792	2.01	2.01	acnebn_



Example of gprof output (2)

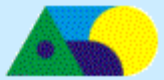
```
                     Call graph (explanation follows)

index % time    self  children    called     name
                0.00   74.78       1/1          main [2]
[1]     92.4    0.00   74.78       1          cnt0_ [1]
                0.00   73.72       1/1            cnt1_ [3]
                0.00    1.05       1/1            su0yomb_ [66]
                0.00    0.02       1/1            su0yoma_ [252]
                0.00    0.00       2/3906         gstats_ [356]
                0.00    0.00       1/1            nmiclose_ [464]
                0.00    0.00       1/2            ppflush_ [437]
                0.00    0.00       1/2            ppclose_ [436]
                0.00    0.00       1/1            ppstat_ [465]
                0.00    0.00       1/1            matclose_ [461]
                0.00    0.00       1/40           user_clock_ [389]
                0.00    0.00       1/15           opdis_ [402]
                0.00    0.00       1/1            stats_output_ [468]
```

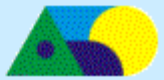
MPI profiling

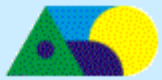
- Purpose: Time spent in message passing which could become significant



Advanced profiling

- Using hardware counters availiable on most modern processors
- Hardware registers for differt kinds of processor events like: floating point operations (FLOPS), cache misses, memory conflicts
- The information could be used in optimization procedure, which is getting more and more important



Conclusions

- Set of tools exist for debugging and profiling of the code
- Highly recomended use of this tools for the code as complex as ALADIN
- Sometimes more timeconsuming to learn the tools than to solve the problem “by hand”
- There are many tools for profiling the CPU consumtion usage but not that many for profiling the memory (stack) usage