Hybrid Parallelization II in SOLPS-ITER

Gaurav Saxena (HLST)
David Vicente Dorca (Head, User Support)

[Barcelona Supercomputing Center, Spain]

EUROfusion Meeting | 27th Oct. 2026

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

Presentation Code COLORS

RGB Presentation Color Code



Bad / Alarm



Good / Improvement



Keyword / Code



File name (can ignore)

What is SOLPS-ITER ?

SOLPS- ITER

- **S**crape-**O**ff **L**ayer **P**lasma **S**imulation (boundary plasma)
- ITER = International Thermonuclear Experimental Reactor
- Fortran 77 (fixed form), Fortran 90
- Monte Carlo code **Eirene** (MPI parallelized + now OpenMP)
- **B2.5** Plasma Fluid solver (OpenMP parallelized, No MPI)

Brief Recap

- **Mid 2023 to 2024 end** - *B2.5 OpenMP performance*
  - Compiler performance options ✅
  - Vectorization ✅
  - Function restructuring ✅
  - Efficient parallelization (initialization, OpenMP optimizations) ✅
  - Porting at new ITER nodes
- **2025** - *Hybrid parallelization Eirene - I*
  - Run-time Error corrections ✅
  - Deadlocks ✅
  - Single thread Vs Serial performance difference ❌ 😳
- **2026** - *Hybrid parallelization Eirene - II*
  - Focus: Single thread Vs Serial performance difference 🔎
  - Others

Focus🔎: The problem from last year

On-going work - Functions with high time variation

Both have 1 B2.5 thread

(A) Eirene 1 MPI process (serial)

(B) Eirene 1 MPI process (with 1 OpenMP thread)

	(A)	(B)
eirene_collide	0.959s	10.439s
eirene_fpath	20.126s	30.564s
eirene_update	18.578s	40.502s
eirene_switch_partinfo	2.611s	10.071s
eirene_timer_triangle_mesh	6.202s	11.929s
eirene_time_to_standard_surface	1.070s	4.578s
eirene_uptbgk	1.140s	5.362s
eirene_uptcop	1.189s	5.743s
eirene_veloel	0.901s	5.038s
h1rn	1.330s	4.771s
eirene_escape	0.250s	1.210s
eirene_folneut	4.079s	6.630s
eirene_locat1	0.430s	2.040s
eirene_reflc1	0.460s	2.190s
eirene_sputr1	0.320s	1.160s
eirene_stdcol	0.060s	0.920s
eirene_stdcol_ass	0.010s	0.330s
eirene_stdcol_x_rad	0.120s	0.830s
eirene_stdnor	0.010s	0.140s
eirene_timea1	0.241s	1.380s
eirene_timer	0.450s	6.699s
eirene_timet	0.561s	2.330s
ncio_px_sync	0.010s	0.861s
__svml_i64rem8_z0	14.117s	11.969s
_f90_reduction_final_strided	0.010s	1.442s
_f90_reduction_init_array	0.020s	1.198s
atan2	0.010s	0.210s

Investigating time difference between serial and OpenMP run with single thread.

Measuring cost of No OpenMP Vs OpenMP

Why the microprogram ?

1. Program pattern analogous to SOLPS-ITER
   a. B2.5 calls threaded Eirene several times - *PATTERN 1*
   b. Eirene performs work inside parallel region - *PATTERN 2*

The microprogram

1. Prevent in-lining of function
2. Use different variable to prevent caching
3. export OMP_NUM_THREADS=1
4. icx -qopenmp -O2 prog.c -o prog.exe
5. Record times for different values of N

```
7   // Prevent inlining
8   __attribute__((noinline))
9   void work(int *x)
10  {
11      *x += 1;
12  }
13
14  int main(void) {
15      int x = 0, y = 0;
16      long double t0, t1;
17      //-------------------------CASE-1: NO OPENMP
18      t0 = omp_get_wtime();
19      for (int i = 0; i < N; i++)
20      {
21          work(&x);
22      }
23      t1 = omp_get_wtime();
24      printf("No OpenMP: %Lf seconds\n", t1 - t0);
25      //-------------------------CASE-2: OPENMP
26      t0 = omp_get_wtime();
27      for (int i = 0; i < N; i++)
28      {
29          #pragma omp parallel
30          {
31              work(&y);
32          }
33      }
34      t1 = omp_get_wtime();
35      printf("OpenMP (1 thread): %Lf seconds\n", t1 - t0);
36      return 0;
37  }
```

Time measurement

N	NO-OPENMP (Serial)	OPENMP (1 thread)
1	0.000002	0.008006
10	0.000002	0.007641
100	0.000004	0.007345
1000	0.000002	0.007560
10,000	0.000001	0.008200
100,000	0.000001	0.015586
1000,000	0.000002	0.090690
10,000,000	0.000002	0.838254

Variation: Moving for loop inside OpenMP parallel

Second pattern in Eirene

1. Work is done in for loop inside OpenMP parallel region.
2. More efficient than pattern 1.

```
7   // Prevent inlining
8   __attribute__((noinline))
9   void work(int *x)
10  {
11      *x += 1;
12  }
13
14  int main(void) {
15      int x = 0, y = 0;
16      long double t0, t1;
17      //--------------------------CASE-1: NO OPENMP
18      t0 = omp_get_wtime();
19      for (int i = 0; i < N; i++)
20      {
21          work(&x);
22      }
23      t1 = omp_get_wtime();
24      printf("No OpenMP: %Lf seconds\n", t1 - t0);
25      //--------------------------CASE-2: OPENMP
26      t0 = omp_get_wtime();
27
28          #pragma omp parallel
29          {
30              for (int i = 0; i < N; i++)
31              {
32                  work(&y);
33              }
34          }
35
36      t1 = omp_get_wtime();
37      printf("OpenMP (1 thread): %Lf seconds\n", t1 - t0);
38      return 0;
39  }
```

Time measurement: NO-OPENMP Vs OPENMP

N	NO-OPENMP (Serial)	OPENMP (1 thread)
1	0.000002	0.008219
10	0.000001	0.007258
100	0.000001	0.007463
1000	0.000002	0.007353
10000	0.000001	0.007062
100000	0.000001	0.007036
1000000	0.000002	0.008775
10000000	0.000002	0.016902

Compiler Optimization Levels

Optimization Level (fixed N=1e7)	NO-OPENMP (Serial)	OPENMP (1 thread)
-O0	0.015756	0.020070
-O1	0.000002	0.021349
-O2	0.000001	0.017163
-O3	0.000001	0.021521

```
7   // Prevent inlining
8   __attribute__((noinline))
9   void work(int *x)
10  {
11      *x += 1;
12  }
13
14  int main(void) {
15      int x = 0, y = 0;
16      long double t0, t1;
17      //-------------------------CASE-1: NO OPENMP
18      t0 = omp_get_wtime();
19      for (int i = 0; i < N; i++)
20      {
21          work(&x);
22      }
23      t1 = omp_get_wtime();
24      printf("No OpenMP: %Lf seconds\n", t1 - t0);
25      //-------------------------CASE-2: OPENMP
26      t0 = omp_get_wtime();
27
28          #pragma omp parallel
29          {
30              for (int i = 0; i < N; i++)
31              {
32                  work(&y);
33              }
34          }
35
36      t1 = omp_get_wtime();
37      printf("OpenMP (1 thread): %Lf seconds\n", t1 - t0);
38      return 0;
39  }
```

OpenMP default(shared) → private

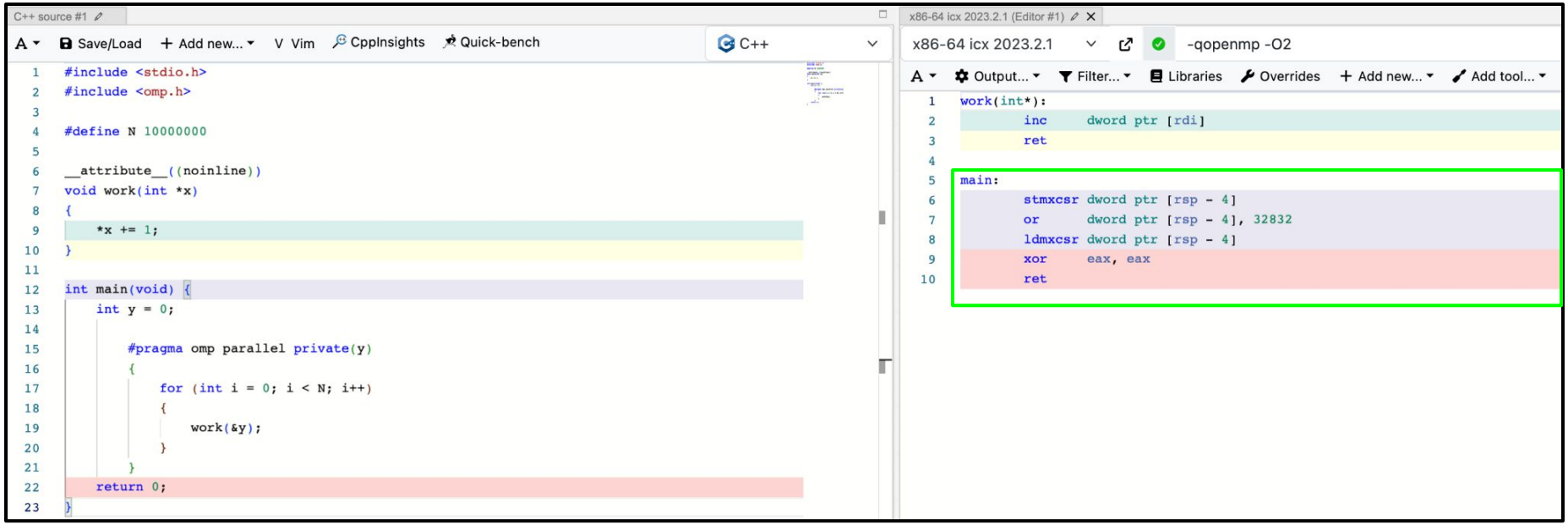
- By default, everything is shared in the OpenMP parallel region.
- What if we declare variable ‘y’ as private ?

Optimization Level (fixed N=1e7)	NO-OPENMP (Serial)	OPENMP (1 thread)
-O0	0.014410	0.021255
-O1	0.000001	0.007807
-O2	0.000002	0.000001
-O3	0.000001	0.000000

```
7   // Prevent inlining
8   __attribute__((noinline))
9   void work(int *x)
10  {
11      *x += 1;
12  }
13
14  int main(void) {
15      int x = 0, y = 0;
16      long double t0, t1;
17      //--------------------------CASE-1: NO OPENMP
18      t0 = omp_get_wtime();
19      for (int i = 0; i < N; i++)
20      {
21          work(&x);
22      }
23      t1 = omp_get_wtime();
24      printf("No OpenMP: %Lf seconds\n", t1 - t0);
25      //--------------------------CASE-2: OPENMP
26      t0 = omp_get_wtime();
27
28          #pragma omp parallel private(y)
29          {
30              for (int i = 0; i < N; i++)
31              {
32                  work(&y);
33              }
34          }
35
36      t1 = omp_get_wtime();
37      printf("OpenMP (1 thread): %Lf seconds\n", t1
38      return 0;
```

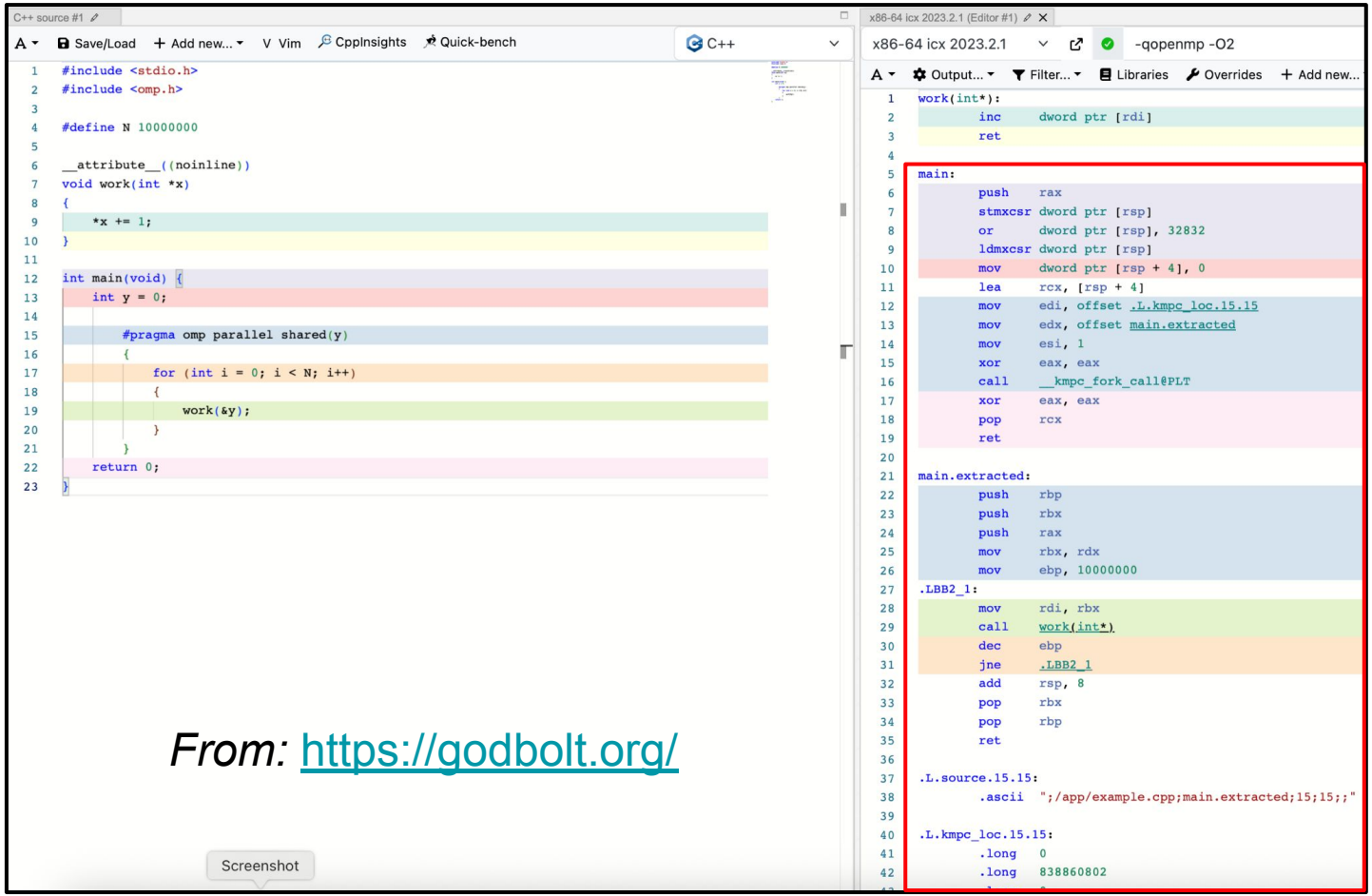
Podcasts

Assembly code inspection: “private”



From: https://godbolt.org/

Assembly code inspection: “shared”



From: https://godbolt.org/

The cost of threadprivate variables

Experiment: Access normal Vs threadprivate

```f90
module data_mod
    implicit none

    integer(kind=8), save :: tp1, tp2, tp3
    integer(kind=8), save :: normal1, normal2, normal3

!$omp threadprivate(tp1, tp2, tp3)

end module data_mod
```

data_mod.f90

- In main.f90 (not shown)
  - Call normal_access() & threadprivate_access() n (~10^9 times)
  - Measure time & ratio

```f90
subroutine normal_access()
    use data_mod
    implicit none

    normal1 = normal1 + 1
    normal2 = normal2 + 2
    normal3 = normal3 + 3

end subroutine normal_access


subroutine threadprivate_access()
    use data_mod
    implicit none

    tp1 = tp1 + 1
    tp2 = tp2 + 2
    tp3 = tp3 + 3

end subroutine threadprivate_access
```

routines.f90

Comparison times: normal Vs threadprivate

Compiler (-O2 -qopenmp)	Normal access (sec)	threadprivate access (sec)	threadprivate/Normal
ifort 2021.10.0	2.73	7.02	2.57x
ifort 2021.11.0	2.73	7.03	2.57x
ifort 2021.13.0	2.73	6.69	2.44x
ifx 2025.1.1	1.07	12.66	11.79x
ifx 2025.2.0	1.03	12.51	12.14x

Understanding the overhead - Macroscopic view

1. Dramatic difference in amount of work (i.e. instructions) b/w normal and threadprivate subroutine access/execution.
2. Normal execution: 3 read-modify-write instructions, address of variables fixed relative to Instruction Pointer (IP), hence no address discovery.
3. Threadprivate execution: First query thread ID, load threadprivate cache, 2 cases: (1) Cache exists (FAST PATH): compute cache slot according to current thread (2) Cache doesn't exist (SLOW PATH): Query OpenMP run-time to load thread-specific address.
4. Point (3) above is repeated for EVERY threadprivate variable.

NOTE: The executable must handle the general case of multiple threads. Setting OMP_NUM_THREADS=1 does not convert threadprivate variables into ordinary globals.

“threadprivate variable” caches Vs “thread” caches

```
GTID = current thread

ptr1 = tp1_cache[GTID]
ptr2 = tp2_cache[GTID]
ptr3 = tp3_cache[GTID]

*ptr1 += 1
*ptr2 += 2
*ptr3 += 3
```

```
base = thread_data_cache[GTID]

base→tp1 += 1
base→tp2 += 2
base→tp3 += 3
```

Solution

1. Use the flag -qopenmp-threadprivate=compat[1]
2. Controls type of threadprivate implementation. Two types:
   a. legacy: Use legacy threadprivate Intel OpenMP implementations
   b. compat: Use “compatibility” threadprivate implementation based on applying thread-local attribute to each threadprivate variable (Used by Microsoft / GNU)
3. “legacy” mode default in ifort, “compat” mode default in ifx (*Web-link*)

[1] Courtesy my colleague *Ziyad AlBanoby*

Timing comparison (1) mini test case (2) SOLPS-ITER[2]

Normal Access	threadprivate access	threadprivate access (-qopenmp-threadprivate=compat)
2.71 sec	7.06 sec	1.32 sec

1. B2.5-Eirene OpenMP
2. ITER_2360 CI test case
3. 10 timesteps, coupled
4. H+He: 5 fluids, 8 strata:

Case	Intel compiler (2023.2.1)	Gfortran compiler (13.2.0)
Serial	3m42.253s	3m24.511s
1 thread	3m35.940s	3m43.500s
2 threads	2m19.262s	2m0.441s
3 threads	2m54.984s	1m36.058s
4 threads	1m36.665s	1m25.123s
6 threads	1m5.954s	1m1.206s
8 threads	1m26.853s	1m15.572s
12 threads	2m27.531s	1m35.017s

[2]Courtesey *Xavier Bonnin*

Others:
Random Seed Generator in Eirene

Hot-spot Subroutine h1rnin() in eirmod_h1rnm.F

Function / Call Stack	CPU Time ▼	Function (Full)
h1rnin	83.934s	h1rnin
eirene_fpath	71.814s	eirene_fpath
eirene_update	63.183s	eirene_update
eirene_timer_triangle_mesh	7.530s	eirene_timer_triangle_mesh
eirene_uptusr	6.549s	eirene_uptusr
eirene_uptcop	4.920s	eirene_uptcop
eirene_uptbgk	4.751s	eirene_uptbgk
ma30adcopy	4.650s	ma30adcopy
eirene_inttal	4.433s	eirene_inttal
eirene_folneut	4.230s	eirene_folneut

224	DO 300 II= 1, 97	0.0%	0.090s
225	S= 0.	0.0%	0.060s
226	T= 0.5		
227	DO 250 JJ= 1,24	1.2%	3.830s
228	M = MOD(MOD(I*J,179)*K, 179)	8.6%	26.507s
229	I = J	1.0%	3.044s
230	J = K		
231	K = M	0.1%	0.420s
232	L = MOD(53*L+1, 169)	0.9%	2.869s
233	IF (MOD(L*M,64) .GE. 32) S = S + T	11.2%	34.548s
234	T = 0.5*T	3.5%	10.776s
235	250 CONTINUE	0.3%	0.880s
236	U(II) = S	0.2%	0.480s
237	300 CONTINUE	0.0%	0.100s

Mathematical manipulation - 1

For any $a >= 0$, $a \% 2^p = IAND(a, 2^p-1)$

Say, $a = 2^p.q+r$ then $a - 2^p.q = r$

1. $0 =< r <= 2^p - 1$, thus all bits in r are in position ‘p’ or ‘below’.
2. $2^p.q$ shifts bits in q by ‘p’ bits, thus all bits in q are at position (p+1) and ‘above’.
3. The only case where $a - 2^p.q = r$ is possible i.e. LHS should not have any bits in positions (p+1) and above as RHS has only bits in positions ‘p’ and below is when we *clear all bit positions (p+1) and above in ‘a’*.
4. Also $a - 2^p.q$ operates only on bit positions (p+1) and above, leaving bits at position ‘p’ and below *unchanged*.
5. Combining the above: $a \% 2^p$ = Bitwise-AND(a, 2^p-1) as 2^p-1 has 0 bits in positions (p+1) and above [i.e. CLEARS the bits] and all 1’s in positions ‘p’ and below [thus, ANDing leaves bits of ‘a’ unchanged]

Mathematical manipulation - 2

((a % m) * b) % m) = (a*b) % m

1. Let a = m.q + r then r = a - m.q
2. Substituting ‘r’ in a % m = r we get, r * b % m = ((a - m.q) * b) % m

   => (a * b - m.q.b) % m = (a * b) % m - 0 as m perfectly divides m.q.b

   Hence proved.

Look-up table optimization

1. The value of T=0.5_DP outside the loop
2. The value of T is halved for JJ=1, 24 i.e. T = T * 0.5
3. Create look-up table:

```
REAL(DP), PARAMETER :: TWOS(24) = (/
& 0.5_DP, 0.25_DP, 0.125_DP, 0.0625_DP,
& 0.03125_DP, 0.015625_DP, 0.0078125_DP,
& 0.00390625_DP, 0.001953125_DP, 0.0009765625_DP,
& 0.00048828125_DP, 0.000244140625_DP,
& 0.0001220703125_DP, 0.00006103515625_DP,
& 0.000030517578125_DP, 0.000015258789625_DP,
& 0.000007629394531250_DP, 0.000003814697265625_DP,
& 0.000001907348632812500_DP, 0.000000953674316406250_DP,
& 0.000000476837158203125_DP, 0.000000238418579101562500_DP,
& 0.000000119209289550781250_DP, 0.000000059604644775390625_DP
& /)
```

4. Then use: S = S + TWOS(JJ)

After Optimizations

Function / Call Stack	CPU Time ▼	Function (Full)
eirene_fpath	70.160s	eirene_fpath
eirene_update	64.401s	eirene_update
h1rnin	61.168s	h1rnin
eirene_timer_triangle_mesh	7.601s	eirene_timer_triangle_mesh
eirene_uptusr	5.948s	eirene_uptusr
eirene_uptbgk	4.830s	eirene_uptbgk
eirene_uptcop	4.750s	eirene_uptcop
eirene_inttal	4.460s	eirene_inttal
ma30adcopy	4.289s	ma30adcopy
eirene_folneut	4.160s	eirene_folneut

h1rnin() time reduced by ~ 22 secs, time reduction = 27.12%

236	DO 300 II= 1, 97	0.0%	0.040s
237	S= 0.	0.0%	0.080s
238	cdr T= 0.5		
239	DO 250 JJ= 1,24	2.7%	7.621s
240	cdr M = MOD(MOD(I*J,179)*K, 179)		
241	M = MOD(I*J*K, 179)	6.0%	17.251s
242	I = J	1.1%	3.179s
243	J = K		
244	K = M	0.1%	0.180s
245	L = MOD(53*L+1, 169)	2.5%	7.290s
246	cdr IF (MOD(L*M,64) .GE. 32) S = S + TWOS(JJ)		
247	IF (IAND(L*M,63) .GE. 32) S = S + TWOS(JJ)	6.8%	19.449s
248	cdr T = 0.5*T		
249	250 CONTINUE	1.7%	4.988s
250	U(II) = S	0.2%	0.600s
251	300 CONTINUE	0.0%	0.130s

Should the RNG seed take so much time ?

1. According to *Detlev Reiter* - creator of the RNG, it should not…
2. RNG Seed incorrectly called for every particle not per stratum (not a normal scenario).
3. Mathematical manipulations not incorporated by developer as may not be compatible with legacy systems.

Others: Compiler Optimization Flags in Eirene

Total time SOLPS-ITER by varying Eirene Compile Flags

Flags	(VTune) Elapsed Time
-fp-model source -fpp	123.841 sec
-fp-model source -fpp -O2	124.08
-fp-model source -fpp -O3	122.83
-fp-model source -fpp -xHost	127.20
-fp-model source -fpp -O3 -qopt-zmm-usage=high	121.28
-fp-model source -fpp -O3 -qopt-zmm-usage=high -xCORE-AVX512	131.14
-fp-model source -fpp -O3 -qopt-zmm-usage=high -ipo	Internal compiler error
-fp-model source -fpp -O3 -qopt-zmm-usage=high **-ip**	121.30

🙏 *Acknowledgement* 🙏

1. Many thanks to Xavier Bonnin for answering all our queries, the quick replies, ideas, testing the code promptly & most importantly for his patience and encouragement.
2. Many thanks to Ziyad AlBanoby for the interesting discussions and contributing to the solution of the single OpenMP thread Vs serial execution performance problem.

Thank you !
For any questions:

Gaurav Saxena

(gaurav.saxena@bsc.es)

&

David Vicente Dorca

(david.vicente@bsc.es)

