HPC-oriented Large-scale Code Restructurings with Coccinelle

or, C/C++ refactorings on steroids

Michele MARTONE[1], Julia LAWALL[2]

[1] Leibniz Supercomputing Centre, Garching near Munich, Germany
[2] Inria Paris, France

HPC, Big Data & Data Science devroom at FOSDEM'24
Bruxelles, February 03, 2024

Suppose you have a

- large C simulation code base ($>$ 100kLoC)
- developed for many years, by many people

Lots of loop accesses

E.g.

```
for(i = 0; i < N; ++i)
  {
    S[i].Metals = S[i].Abundance * factor + S[i].DeltaMetals;
    S[i].Abundance = S[i].Metals * P[i].InvMass;
    S[i].DeltaMetals = 0;
  }
```

What could be most obvious optimization (for CPUs)?

```
for(i = 0; i < N; ++i)
  {
    S[i].Metals = S[i].Abundance * factor + S[i].DeltaMetals;
    S[i].Abundance = S[i].Metals * P[i].InvMass;
    S[i].DeltaMetals = 0;
  }
```

What could be most obvious optimization (for CPUs)?

Hint: S[i] are structures

```
for(i = 0; i < N; ++i)
  {
    S[i].Metals = S[i].Abundance * factor + S[i].DeltaMetals;
    S[i].Abundance = S[i].Metals * P[i].InvMass;
    S[i].DeltaMetals = 0;
  }
```

Change AoS into Structure-of-Arrays (SoA)

```
for(i = 0; i < N; ++i)
  {
    S[i].Metals = S[i].Abundance * factor + S[i].DeltaMetals;
    S[i].Abundance = S[i].Metals * P[i].InvMass;
    S[i].DeltaMetals = 0;
  }
```

$\Rightarrow$

```
for(i = 0; i < N; ++i)
  {
    SoA.Metals[i] = S[i].Abundance * factor + S[i].DeltaMetals;
    S[i].Abundance = SoA.Metals[i] * P[i].InvMass;
    S[i].DeltaMetals = 0;
  }
```

Ok but how?

Ok but how?

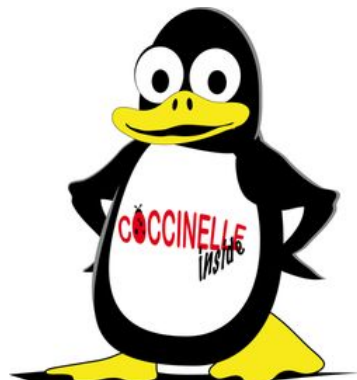
- regexps (`sed`, `awk`, `perl`, `python`...)?

Ok but how?

- regexps (`sed`, `awk`, `perl`, `python`...)?
- a *source-to-source translator*?
  LLVM?
  ROSE?

Ok but how?

- regexps (`sed`, `awk`, `perl`, `python`...)?
- a *source-to-source translator*?
  LLVM?
  Rose?
  Coccinelle!!



Originally, *to identify* and *to eliminate* bugs

AoS to SoA in a Nutshell

```
@@
identifier q =~ "prsr";
identifier s = {P};
identifier i = {i,k};
fresh identifier n = s ## "_" ## q;
@@
- s[i].q
+ n[i]
```

basicaos2soa1.cocci

```
@@ -1,9 +1,9 @@
 struct p_t {
   double prsr, vel;
 };

 p_t P[3];
 int main() {
         int i,j,k;
-        P[i].prsr++;
+        P_prsr[i]++;
 }
```

basicaos2soa1.diff

HPC challenges for Coccinelle

Managing complexity

- GPU *language extensions* (CUDA, HIP, ...)
- modern C++ features (mdspan, SYCL, ...)
- advanced `#pragma` manipulation: OpenMP, ...

New: CUDA GPUs extensions

```
#spatch --smpl-spacing
@@
attribute name __device__;
@@
+ __device__
  int f(void) {
+   const int gthid =
+     blockIdx.x * blockDim.x
+                + threadIdx.x;
    ...
 }

@@
identifier d,b;
@@
  int d;
  int b;
- f();
+ f<<<d,(d+b-1)/b>>>();
```

```
@@ -1,10 +1,13 @@
-int f(void)
+__device__ int f(void)
 {
+  const int gthid =
+      blockIdx.x * blockDim.x
+                 + threadIdx.x;
   { /* loops */ }
 }
 int main(void)
 {
   int ds;
   int bs;
-  f();
+  f<<<ds,(ds+bs-1)/bs>>>();
 }
```

cuda.diff

New: C++23 multi-index operators

```
@multiindex@
symbol a,i,j,k;
@@
- a[i][j][k]
+ a[i, j, k]

@@
symbol b;
@@
- b[...]
+ b[0]
```

mdspan1.cocci

```
@@ -1,8 +1,8 @@
 int main()
 {
         int a[1][1][1];
         int b[1][1][1];
         int i=0,j=0,k=0;
-        a[i][j][k]++;
-        b[i][j][k]++;
+        a[i, j, k]++;
+        b[0][j][k]++;
 }
```

mdspan1.diff

New: decluttering with OpenMP 5.1 pragmas

```
@@
identifier c, f, l;
expression b, k;
type T;
@@

+ const T j = k;
+ #pragma omp unroll partial (j)
  for (T c=0; c
-                 + k - 1
                          < l ; ... )
 {
-        f(b+0);
-        f(b+1);
-        f(b+2);
-        f(b+3);
+        f(b);
 }
```

deunroll1.cocci

```
@@ -1,12 +1,11 @@
 int f(int i){}
 int main()
 {
         const int n = 13;
-        for (int i=0;i+4-1<n;i++)
+        const int j = 4;
+#pragma omp unroll partial (j)
+        for (int i=0;i<n;i++)
         {
-                f(i+0);
-                f(i+1);
-                f(i+2);
-                f(i+3);
+                f(i);
         }
 }
```

deunroll1.diff

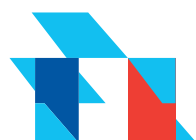
Currently working on

- covering C++ features
- further use cases

Acknowledgements

Work made possible by

- SiVeGCS Project
- BayFrance travel grant



Resources

- new tutorial at deRSE24 in Würzburg, 05-07.03.2024
  https://go.uniwue.de/derse24
- old 247-page training from 2019
  https://www.lrz.de/services/compute/courses/x_lecturenotes/hspc1w19.pdf
- a 6-page use case article
  https://inria.hal.science/hal-03266521
- Coccinelle's GitHub page
  https://github.com/coccinelle/coccinelle