GNU Cauldron

Loop optimizations enabled by data dependence analysis (Graphite):

- Current work:
  - Bootstrap enabled with graphite
  - Make check
  - Remove limit scops (under development)
  - loop fusion

- Future Plans:
  - Short term (6 months):
    - Delinearization
    - Test graphite with all loop flags enabled
    - Compile time (isl timeout)
    - Profile guided isl-timeout
    - Loop scheduling: modulo scheduling, unroll and jam

  - Longer term:
    - Interface with gcc vectorizer
    - Improving/Adding cost functions (e.g., Minimize data traffic)
    - Comparative analysis with poly and improve graphite where poly performs better
    - Benchmarking
    - Splitting BBs with one data-ref
    - Try with -flto

Vectorization:

- benchmark analysis:
    - TSVC: some loops not yet vectorized
    - OpenCV: loops with control flow require code restructuring (jump threading, versioning)
    - SPEC: some loops require fusion, interchange, and distribution
- Improve if-conversion: select code gen based on target:
    - scratch-pad
    - masked load/store
-

TSVC: GCC vs. LLVM

----------Test1----------

```
#define TYPE double
#define ALIGNMENT 32
#define LEN 32000
#define LEN2 256
void set(int* ip, TYPE* s1, TYPE* s2){
  posix_memalign((void **) &xx, ALIGNMENT, LEN*sizeof(TYPE));
  printf("\n");
  for (int i = 0; i < LEN; i = i+5){                    // None vectorize
     ip[i]   = (i+4);
     ip[i+1] = (i+2);
     ip[i+2] = (i);
     ip[i+3] = (i+3);
     ip[i+4] = (i+1);
  }
  for (int i = 0; i < LEN; i++){                        // gcc vectorizes
     indx[i] = (i+1) % 4+1;
  }
  *s1 = 1.0;
  *s2 = 2.0;
}
```

----------Test2----------

```
int s161()
{

//      control flow
//      tests for recognition of loop independent dependences
//      between statements in mutually exclusive regions.

 for (int nl = 0; nl < ntimes/2; nl++) {                // None vectorize
   for (int i = 0; i < LEN-1; ++i) {                    // None vectorize
     if (b[i] < (TYPE)0.) {
       goto L20;
     }
     a[i] = c[i] + d[i] * e[i];
     goto L10;
L20:
     c[i+1] = a[i] + d[i] * d[i];
L10:
     ;
```

```
  }
 }
 return 0;
}
```

----------Test3----------

```
int s171(int inc)
{
  for (int nl = 0; nl < ntimes; nl++) {
      for (int i = 0; i < LEN; i++) {                    // llvm vectorizes
          a[i * inc] += b[i];
      }
  }
  return 0;
}
```

OpenCV example:

```
   for( k = 0, n = 0; k < upper_total/2 && n < lower_total/2; )
      {
         switch( connect_flag )
         {
         case ICV_SINGLE:
            if( upper_run->next->pt.x < lower_run->next->pt.x )
         [...]
         case ICV_CONNECTING_ABOVE:
            if( upper_run->pt.x > lower_run->next->pt.x +1 )
         [...]
         }
      }// k, n
```