Center for High-Performance Computing (ZHR)

Dresden University of Technology

Code Crumpling: A Straight Technique to Improve Loop Performance on Cache Based Systems

Stephan Seidl
seidl@zhr.tu-dresden.de

Abstract

While parallelization has made important progress during the last few years, the sequential performance relatively stagnates, although we have taken the 1 GHz hurdle. Moreover, we have to put tens of processors into action to be competitive with vector machines. Once an application is analyzed by means of performance tools like Vampir, the user is typically faced with some loops which consume most of the CPU time. To improve loop performance, the present paper assumes that it is just enough in many cases to apply a generalized blocking technique with respect to the size of one primary cache line, or two of them occasionally. Its primary goal is to maximize the number of user operations divided by the expected value for the number of drawn primary cache lines. Another experience is that the inner loop body has to be well-stuffed and has to have as little memory write accesses as possible. More than 50 Fortran/C example loops have been investigated so far, and essential results are presented. The performance in speed correlates with results which we have seen for routines in the BLAS libraries on different platforms. Finally, because of its clear goals, this procedure can also easily be taught.

Merit function to control code transformations

$$f = \frac{<\#\ \text{user operations}>}{<\#\ \text{drawn primary cache lines}>}$$

Studying matrix multiplication

Initial code

```
  for (i = 0; i < n; i += 1) {
  for (j = 0; j < n; j += 1) {
  for (k = 0; k < n; k += 1) {
  c[i][j] += a[i][k] * b[k][j]; /* u1=2 */
/* w1=0       w2=1/m     w3=1 */
  } } }
```

$$f = \frac{2\,m}{m+1}$$

m ... # of operands per cache line

First try

```
  for (i = 0; i < n; i += 1) {
  for (j = 0; j < n; j += 1) {
  for (k = 0; k < n; k += 2) {
  c[i][j] += a[i][k + 0] * b[k + 0][j]; /* u1=2 */
/* w1=0          w2=1/m        w3=1 */
  c[i][j] += a[i][k + 1] * b[k + 1][j]; /* u2=2 */
/* w4=0          w5=1/m        w6=1 */
  } } }
```

$$f = \frac{2\,m}{m+1}$$

Unrolled j

```
  for (i = 0; i < n; i += 1) {
  for (j = 0; j < n; j += 2) {
  for (k = 0; k < n; k += 1) {
  c[i][j + 0] += a[i][k] * b[k][j + 0]; /* u1=2 */
   /* w1=0        w2=1/m       w3=1    */
  c[i][j + 1] += a[i][k] * b[k][j + 1]; /* u2=2 */
   /* w4=0        w5=0         w6=1/m */
  } } }
```

$$f = \frac{2\,q\,m}{m+q}$$

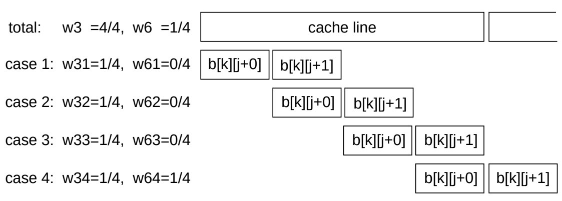
q ... unroll factor

Unrolled i, j

```
  for (i = 0; i < n; i += 2) {
  for (j = 0; j < n; j += 2) {
  for (k = 0; k < n; k += 1) {
  c[i + 0][j + 0] += a[i + 0][k] * b[k][j + 0]; /* u1=2 */
     /* w1=0            w2=1/m         w3=1    */
  c[i + 0][j + 1] += a[i + 0][k] * b[k][j + 1]; /* u2=2 */
     /* w4=0            w5=0           w6=1/m */
  c[i + 1][j + 0] += a[i + 1][k] * b[k][j + 0]; /* u3=2 */
     /* w7=0            w8=1/m         w9=0    */
  c[i + 1][j + 1] += a[i + 1][k] * b[k][j + 1]; /* u4=2 */
     /* wa=0            wb=0           wc=0    */
  } } }
```

$$f = \frac{2\,q^2\,m}{m+2\,q-1}$$

Unrolled j, different alignment situations



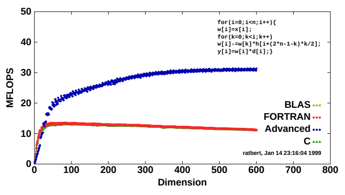
A level-2 example

```
for (j0 = 0; j0 < j1; j0++)
  f0 += *(p0 + j0) * *(p1 + ((j2 - j0) * j0 >> 1));
```

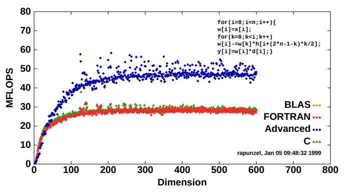
initial inner loop

```
for (j3 = 0; j3 < j4; j3++) {
  f0 += *(p2 + j3) * *(p3 + ((j5 - j3) * j3 >> 1) + 0);
  f1 += *(p2 + j3) * *(p3 + ((j5 - j3) * j3 >> 1) + 1);
  f2 += *(p2 + j3) * *(p3 + ((j5 - j3) * j3 >> 1) + 2);
  f3 += *(p2 + j3) * *(p3 + ((j5 - j3) * j3 >> 1) + 3);
  f4 += *(p2 + j3) * *(p3 + ((j5 - j3) * j3 >> 1) + 4);
  f5 += *(p2 + j3) * *(p3 + ((j5 - j3) * j3 >> 1) + 5);
  f6 += *(p2 + j3) * *(p3 + ((j5 - j3) * j3 >> 1) + 6);
  f7 += *(p2 + j3) * *(p3 + ((j5 - j3) * j3 >> 1) + 7);
  }
```

inner loop after optimization, theoretical acceleration 3.33 (m=4, q=8)



Level-2 type performance on T3E-600 with streams off



Level-2 type performance on 195-MHz Origin2000

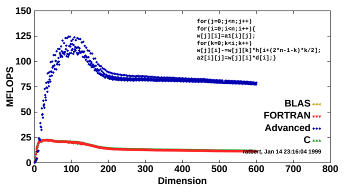
A level-3 example

```
for (j0 = 0; j0 < j1; j0++)
  f0 += *(p0 + j0) * *(p1 + ((j2 - j0) * j0 >> 1));
```

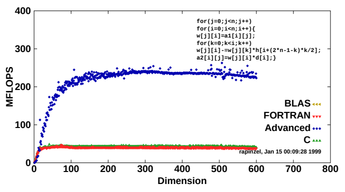
initial inner loop

```
for (j6 = 0; j6 < j7; j6++) {
  f0 += *(p4 + n * 0 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 0);
  f1 += *(p4 + n * 1 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 0);
  f2 += *(p4 + n * 2 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 0);
  f3 += *(p4 + n * 3 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 0);
  f4 += *(p4 + n * 0 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 1);
  f5 += *(p4 + n * 1 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 1);
  f6 += *(p4 + n * 2 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 1);
  f7 += *(p4 + n * 3 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 1);
  f8 += *(p4 + n * 0 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 2);
  f9 += *(p4 + n * 1 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 2);
  fa += *(p4 + n * 2 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 2);
  fb += *(p4 + n * 3 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 2);
  fc += *(p4 + n * 0 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 3);
  fd += *(p4 + n * 1 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 3);
  fe += *(p4 + n * 2 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 3);
  ff += *(p4 + n * 3 + j6) * *(p5 + ((j8 - j6) * j6 >> 1) + 3);
  }
```

inner loop after optimization, theoretical acceleration 7.27 (m=4, q=4)



Level-3 type performance on T3E-600 with streams off



Level-3 type performance on 195-MHz Origin2000