Simulation and High-Performance Computing

Part 12: OpenMP Tasks

Steffen Börm

Christian-Albrechts-Universität zu Kiel

October 5th, 2020

Synchronization so far

Barriers: All threads have to reach a barrier before they can move on.

Critical sections: Only one thread at a time can execute a critical section.

Locks: Once a lock is set, all other threads trying to set it have to wait until it is unset again.

OpenMP Tasks

Idea: Split a program into tasks that can be performed in parallel.

- Each task has its own instruction pointer, registers, and stack.
- Every task is assigned to one of the threads in the current team for execution.
- The execution of a task can be suspended and later resumed by the system to execute another task.
- Tasks can create further tasks and wait for their completion.
- Tasks can have data dependencies.

Task creation

```
#pragma omp parallel
#pragma omp single
{
  #pragma omp task private(i)
  for(i=0; i<n/2; i++)
    x[i] += alpha * y[i];

  #pragma omp task private(i)
  for(i=n/2; i<n; i++)
    x[i] += alpha * y[i];
}
```

- *#pragma omp task* creates a task to execute the following structured block, in this case a `for` loop.
- *#pragma omp single* is used to ensure that each task is created only once.

Example: Forward transformation

Goal: Given a cluster tree, we want to compute

$$z_{\sigma,\nu} := \sum_{j \in \hat{\sigma}} \ell_{\sigma,\nu}(y_j)\, m_j$$

for all clusters. If σ has children, we have

$$z_{\sigma,\nu} = \sum_{\sigma' \in \mathrm{chil}(\sigma)} \sum_{\nu' \in M} \ell_{\sigma,\nu}(\xi_{\sigma',\nu'})\, z_{\sigma',\nu'}.$$

Task parallelization: We have to ensure that all childrens' coefficients have been computed before we can compute $z_{\sigma,\nu}$.

Forward transformation with tasks

```
void
forward(cluster *sigma)
{
  if(sigma->children > 0) {
    for(i=0; i<sigma->children; i++)
      #pragma omp task firstprivate(sigma,i)
      forward(sigma->chil[i]);

    #pragma omp taskwait

    /* compute coefficients */
  }
  else {
    /* compute coefficients */
  }
}
```

Task dependencies

```
int i = 0, j = 0, k = 0;

#pragma omp task depend(out:i)
i++;

#pragma omp task depend(out:j)
j--;

#pragma omp task depend(in:i), depend(inout:k)
k += i;

#pragma omp task depend(inout:j), depend(in:i,k)
j -= i + k;
```

- The first two tasks can be executed in parallel.
- The third task has to wait for the first.
- The fourth task has to wait for all others.

Block matrices

Goal: Linear algebra parallelized with tasks.

Problem: Task creation takes time, so we should not create a task for every single arithmetic operation.

Approach: Split matrices into $m \times m$ blocks of size $\ell \times \ell$ with $n = m\ell$.

$$A = \begin{pmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mm} \end{pmatrix}, \qquad A_{ij} \in \mathbb{R}^{\ell \times \ell} \text{ for all } i, j \in [1 : m].$$

The blocks should be

- small enough to fit into the processor's cache, but
- large enough to keep the number of tasks manageable.

Remark: Block matrices and caches

Cache: Since access to the main memory is slow, processors have a small amount of fast cache memory that contains copies of recently used parts of the main memory.

Idea: If we need to access the same coefficients repeatedly, they should be kept in the cache if possible.

Eligible operations: The matrix multiplication, LU factorization, as well as forward and backward solves for n right-hand sides all perform $\sim n^3$ operations on $\sim n^2$ coefficients. $\rightarrow$ Cache should be useful.

BLAS Level 3: Matrix multiplication $C \leftarrow \alpha A B + \beta C$.

```
void gemm(bool transA, bool transB,
          int rows, int cols, int k,
          real alpha, const real *A, int ldA,
                      const real *B, int ldB,
          real beta,  real *C, int ldC);
```

Block matrix multiplication

Goal: Compute $C = AB$ with $A, B, C \in \mathbb{R}^{n \times n}$.

Block representation:

$$\begin{pmatrix} C_{11} & \cdots & C_{1m} \\ \vdots & \ddots & \vdots \\ C_{m1} & \cdots & C_{mm} \end{pmatrix} = \begin{pmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mm} \end{pmatrix} \begin{pmatrix} B_{11} & \cdots & B_{1m} \\ \vdots & \ddots & \vdots \\ B_{m1} & \cdots & B_{mm} \end{pmatrix}.$$

Block matrix multiplication

Goal: Compute $C = AB$ with $A, B, C \in \mathbb{R}^{n \times n}$.

Block representation:

$$\begin{pmatrix} C_{11} & \cdots & C_{1m} \\ \vdots & \ddots & \vdots \\ C_{m1} & \cdots & C_{mm} \end{pmatrix} = \begin{pmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mm} \end{pmatrix} \begin{pmatrix} B_{11} & \cdots & B_{1m} \\ \vdots & \ddots & \vdots \\ B_{m1} & \cdots & B_{mm} \end{pmatrix}.$$

Block matrix multiplication

Goal: Compute $C = A\,B$ with $A, B, C \in \mathbb{R}^{n \times n}$.

Block representation:

$$\begin{pmatrix} C_{11} & \cdots & C_{1m} \\ \vdots & \ddots & \vdots \\ C_{m1} & \cdots & C_{mm} \end{pmatrix} = \begin{pmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mm} \end{pmatrix} \begin{pmatrix} B_{11} & \cdots & B_{1m} \\ \vdots & \ddots & \vdots \\ B_{m1} & \cdots & B_{mm} \end{pmatrix}.$$

Block matrix multiplication

Goal: Compute $C = A\,B$ with $A, B, C \in \mathbb{R}^{n\times n}$.

Block representation:

$$\begin{pmatrix} C_{11} & \cdots & C_{1m} \\ \vdots & \ddots & \vdots \\ C_{m1} & \cdots & C_{mm} \end{pmatrix} = \begin{pmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mm} \end{pmatrix} \begin{pmatrix} B_{11} & \cdots & B_{1m} \\ \vdots & \ddots & \vdots \\ B_{m1} & \cdots & B_{mm} \end{pmatrix}.$$

Block matrix multiplication

Goal: Compute $C = AB$ with $A, B, C \in \mathbb{R}^{n \times n}$.

Block representation:

$$\begin{pmatrix} C_{11} & \cdots & C_{1m} \\ \vdots & \ddots & \vdots \\ C_{m1} & \cdots & C_{mm} \end{pmatrix} = \begin{pmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mm} \end{pmatrix} \begin{pmatrix} B_{11} & \cdots & B_{1m} \\ \vdots & \ddots & \vdots \\ B_{m1} & \cdots & B_{mm} \end{pmatrix}.$$

Block matrix multiplication

Goal: Compute $C = AB$ with $A, B, C \in \mathbb{R}^{n \times n}$.

Block representation:

$$\begin{pmatrix} C_{11} & \cdots & C_{1m} \\ \vdots & \ddots & \vdots \\ C_{m1} & \cdots & C_{mm} \end{pmatrix} = \begin{pmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mm} \end{pmatrix} \begin{pmatrix} B_{11} & \cdots & B_{1m} \\ \vdots & \ddots & \vdots \\ B_{m1} & \cdots & B_{mm} \end{pmatrix}.$$

Result: We have to compute

$$C_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj} \qquad \text{for all } i, j \in [1 : m].$$

Matrix multiplication with tasks

```
for(k=0; k<m; k++)
  for(j=0; j<m; j++)
    for(i=0; i<m; i++)
      #pragma omp task firstprivate(i,j,k), \
              depend(in:A[i*m + k*m*ldA], B[k*m + j*m*ldB]), \
              depend(inout:C[i*m + j*m*ldC])
      gemm(false, false, l, l, l, 1.0,
           A + i*m + k*m*ldA, ldA, B + k*m + j*m*ldB, ldB,
           C + i*m + j*m*ldC);
```

- For the dependencies, the upper left coefficient of a matrix block represents the entire block, e.g., `C[i*m + j*m*ldC]` represents C_{ij}.
- We assume that A, B, C, and ℓ remain unchanged in this procedure. For a general implementation, they would have to be included in the `firstprivate` clause.

Block LU factorization

Goal: Find lower triangular L and upper triangular U with $A = L\,U$.

Block representation with $A_{11}, L_{11}, U_{11} \in \mathbb{R}^{\ell \times \ell}$:

$$\begin{pmatrix} L_{11} & \\ L_{*1} & L_{**} \end{pmatrix} \begin{pmatrix} U_{11} & U_{1*} \\ & U_{**} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{1*} \\ A_{*1} & A_{**} \end{pmatrix}$$

Result: We have to solve the equations

$$\begin{aligned} L_{11} U_{11} &= A_{11}, \\ L_{*1} U_{11} = A_{*1}, &\qquad L_{11} U_{1*} = A_{1*}, \\ L_{**} U_{**} &= A_{**} - L_{*1} U_{1*}. \end{aligned}$$

The first is an LU factorization, the last a matrix multiplication.
The second and third can be solved by forward substitution.

Forward substitution $LX = Y$

Goal: Solve $LX = Y$ with L lower left triangular.

Approach: Split into submatrices.

$$\begin{pmatrix} \ell_{11} & \\ L_{*1} & L_{**} \end{pmatrix} \begin{pmatrix} X_1 \\ X_* \end{pmatrix} = \begin{pmatrix} Y_1 \\ Y_* \end{pmatrix}$$

Result: Solve $\ell_{11} X_1 = Y_1$ and $L_{**} X_* = Y_* - L_{*1} X_1$.

Implementation with BLAS: Overwrite Y with solution X.

```
for(k=0; k<l; k++)
  ger(l-k-1, m, -1.0,
      L+(k+1)+k*ldL, 1, Y+k, ldY,
      Y+(k+1), ldY);
```

Forward substitution $XU = Y$

Goal: Solve $XU = Y$ with U upper right triangular.

Approach: Split into submatrices.

$$\begin{pmatrix} X_1 & X_* \end{pmatrix} \begin{pmatrix} u_{11} & U_{1*} \\ & U_{**} \end{pmatrix} = \begin{pmatrix} Y_1 & Y_* \end{pmatrix}$$

Result: Solve $X_1 u_{11} = Y_1$ and $X_* U_{**} = Y_* - X_1 U_{1*}$.

Implementation with BLAS:

```
for(k=0; k<l; k++) {
  scal(m, 1.0/U[k+k*ldU], Y+k*ldY, 1);
  ger(m, l-k-1, -1.0,
      Y+k*ldY, 1, U+k+(k+1)*ldU, ldU,
      Y+(k+1)*ldY, ldY);
}
```

LU factorization with tasks, Part 1

```
for(k=0; k<m; k++) {
  #pragma omp task firstprivate(k), \
          depend(inout:A[k*m + k*m*ldA])
  ludecomp(l, A + k*m + k*m*ldA, ldA);

  for(i=k+1; i<m; i++) {
    #pragma omp task firstprivate(i,k), \
            depend(in:A[k*m + k*m*ldA]), \
            depend(inout:A[k*m + i*m*ldA])
    lsolve(l, l, A + k*m + k*m*ldA, ldA, A + k*m + i*m*ldA, ldA);

    #pragma omp task firstprivate(i,k), \
            depend(in:A[k*m + k*m*ldA]), \
            depend(inout:A[i*m + k*m*ldA])
    rsolve(l, l, A + k*m + k*m*ldA, ldA, A + i*m + k*m*ldA, ldA);
  }

  /* ... Update A_{ij} -= L_{ik} U_{kj} ... */
}
```

LU factorization with tasks, Part 2

```
for(k=0; k<m; k++) {
  #pragma omp task firstprivate(k), \
          depend(inout:A[k*m + k*m*ldA])
  ludecomp(l, A + k*m + k*m*ldA, ldA);

  /* ... Compute L_{ik} and U_{kj} ... */

  for(j=k+1; j<m; j++)
    for(i=k+1; i<m; i++)
      #pragma omp task firstprivate(i,j,k), \
              depend(in:A[i*m + k*m*ldA], A[k*m + j*m*ldA]), \
              depend(inout:A[i*m + j*m*ldA])
      gemm(false, false, l, l, l, -1.0,
           A + i*m + k*m*ldA, ldA, A + k*m + j*m*ldA, ldA,
           A + i*m + j*m*ldA, ldA);
}
```

Experiment: Linear algebra with tasks

Approach: Compute products and LU factorization of 4096×4096 matrices on an eight-core processor.

	MM		LU	
Standard	22.98		11.38	
Block, $\ell = 64$	19.23	1.20	6.99	1.63
Tasks, $\ell = 64$	2.69	8.54	1.04	10.94
Block, $\ell = 128$	16.50	1.39	6.03	1.89
Tasks, $\ell = 128$	2.15	10.69	0.81	14.05
Block, $\ell = 256$	13.30	1.73	5.12	2.22
Tasks, $\ell = 256$	1.94	11.85	1.08	10.54

Summary

Tasks can be used to distribute computations among threads.

Data dependencies can be used to synchronize tasks.

Block algorithms allow us to take advantage of caches and split time-consuming computations into tasks.