

# Ct: Data Parallel Programming

```
void swap_csr_double_3x3(csrMatrix &A,
                           const int &rows,
                           const int &cols,
                           const int &nz,
                           const int &nzmax,
                           const int &nzptr,
                           const double * &values,
                           const int &nnz) {
    int i;
    double temp;
    int scratch_i = A.nzptr[0];
    int scratch_j = A.nzptr[0] + nzmax - A.nzptr[1];
    int scratch_nz = scratch_j - scratch_i;
    for (i = 0; i < nzmax; i++) {
        if (nzptr[i] == nzptr[i+1]) { // don't swap across columns
            temp = values[scratch_i];
            values[scratch_i] = values[scratch_j];
            values[scratch_j] = temp;
            scratch_i = scratch_j;
            scratch_j = scratch_j + nzmax - nzptr[i+1];
        }
        else { // swap between columns
            temp = values[scratch_i];
            values[scratch_i] = values[scratch_nz];
            values[scratch_nz] = temp;
            scratch_i = scratch_nz;
            scratch_nz = scratch_nz + nzmax - nzptr[i+1];
        }
    }
}
```

C with OpenMP alone:  
172 lines of code

```
if (!openmp_bound || !omp_is_initialized())
    #pragma omp parallel for private(nnz)
    #pragma omp num_threads(1) firstprivate(rows, cols, nzmax, nzptr, nnz, nz)
    #pragma omp num_threads(1) shared(A)
    #pragma omp num_threads(1) reduction(+:temp)
    for (int i = 0; i < nzmax; i++) {
        if (nzptr[i] == nzptr[i+1]) { // don't swap across columns
            temp = values[omp_get_thread_num() * (nzptr[i+1] - nzptr[i]) + i];
            values[omp_get_thread_num() * (nzptr[i+1] - nzptr[i]) + i] = values[nzptr[i+1] - 1];
            values[nzptr[i+1] - 1] = temp;
        }
        else { // swap between columns
            temp = values[omp_get_thread_num() * (nzptr[i+1] - nzptr[i]) + i];
            values[omp_get_thread_num() * (nzptr[i+1] - nzptr[i]) + i] = values[nzptr[i+1] - 1];
            values[nzptr[i+1] - 1] = temp;
        }
    }
```

**Ct will extend C/C++ by adding new data structures & operators which exploit opportunities for the parallel processing of data**

- Greater performance due to concurrent execution
- Library-like interface compatible with existing programming environment
- Optimizes code at run time for the user's hardware
- Scalable from 1 to  $n$  cores

Ct: 6 lines of code,  
faster, scalable

```
CCtTVEC<double> sparseMatrixVectorProduct(
    ccTVEC<double> A, CCtVEC<int> rowindex,
    CCtVEC<int> cols, CCtVEC<double> v)
{
    ccTVEC expv = ctDistribute(v,cols);
    CCtVEC product = A*expv;
    return ctMultiReduceSum(product, rowindex);
}
```

