The Cilk Plus Extension

José Alves, Rui Brito

Universidade do Minho

Braga, May 2013

Index

What is Cilk Plus

- Intel Cilk Plus is an extension to the C and C++ languages to support task and data parallelism;
- As an extension implemented in the compiler it provides lower overhead than library-only solutions;

Key Features

- Keywords: cilk adds three new keywords to the C/C++ languages to better express task parallelism in an application;
- Reducers: a mechanism to eliminate contention for shared variables;
- #pragma simd: a directive that tells the compiler a loop should be vectorized;
- Array Notations: a language addition to specify data parallelism for arrays or sections of arrays;

Keywords

- cilk_spawn: tells the runtime environment that the statement following the cilk_spawn keyword can be run in parallel with other statements;
- cilk_sync: tells the runtime environment that all children of the spawning block must finish their execution before execution can continue;
- cilk_for: a replacement for the standard C/C++ for loop that lets iterations run in parallel;

cilk_spawn example

```
double fib(double n) {
    if (n < 2)
        return n;

    if (n < 30)
        return seq_fib(n);

    double x = cilk_spawn fib(n-1);
    double y = fib(n-2);
    cilk_sync;
    return x + y;
}
```

cilk_for example

```
float* matrixDot (float *a, float *b, int n) {

float value = 0;
float *res = (float*) malloc(n * n * sizeof(float));

cilk_for (int i = 0; i < n; ++i) {
for(int j = 0; j < n; ++j) {
for(int k = 0; k < n; ++k) {
value += a[i * n + k] * b[k * n + j];
}
res[i * n + j] = value;
value = 0;
}
}


return res;

}
```

More on cilk_for

- Using cilk_for is not the same as spawning each loop iteration.
- The compiler converts the loop body to a function that is called recursively using a divide-and-conquer strategy;

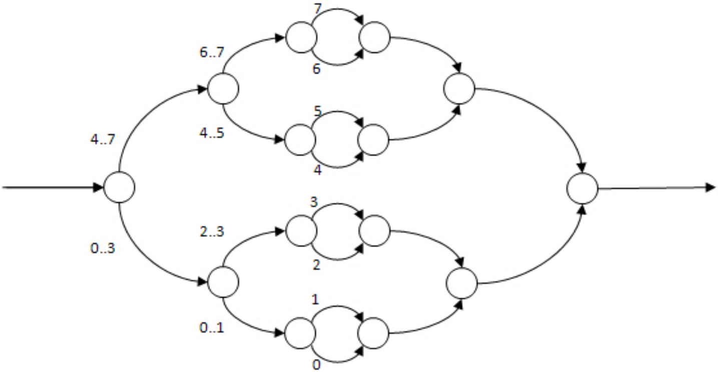

Figure : cilk_for with N=8 iterations

Reducers

```
void reducer_list_test() {
    cilk::reducer_list_append<char> letters_reducer;

    cilk_for(char ch = 'a'; ch <= 'z'; ch++){
        simulated_work();
        letters_reducer.push_back(ch);
    }

    const std::list<char> &letters = letters_reducer.
        get_value();

    for(std::list<char>::const_iterator i = letters.begin();
         i != letters.end(); i++) {
        std::cout << " " << *i;
    }
    std::cout << std::endl;
}
```

Available reducers

- reducer_list_append: Creates a list by adding elements to the back;
- reducer_list_prepend: Creates a list by adding elements to the front;
- reducer_max: Calculates the maximum value of a set of values;
- reducer_max_index: Calculates the maximum value and index of that value of a set of values;
- reducer_min: Calculates the minimum value of a set of values;
- reducer_min_index: Calculates the minimum value and index of that value of a set of values;
- reducer_opadd: Calculates the sum of a set of values;
- reducer_opand: Calculates the binary AND of a set of values;
- reducer_opor: Calculate the binary OR of a set of values;
- reducer_opxor: Calculate the binary XOR of a set of values;
- reducer_string: Accumulates a string using append operations;
- reducer_wstring: Accumulates a "wide" string using append operations;
- reducer_ostream: An output stream that can be written in parallel;
- reducer_ostream: An output stream that can be written in parallel;

Array Notation

- An array section assignment is a parallel operation that modifies every element of the array section on the left-hand side;
- Makes uses of vectorization techniques;
- Explicit high-level mechanism for expressing SIMD instructions (explicit #pragma simd);

Examples

```
// Copy elements 10->19 in A to elements 0->9 in B.
B[0:10] = A[10:10];
// Transpose row 0, columns 0-9 of A, into column 0, rows 0-9 of B.
B[0:10][0] = A[0][0:10];
// Copy the specified array section in the 2nd and 3rd dimensions of A into the 1st and 4th dimensions of B.
B[0:10][0][0][0:5] = A[3][0:10][0:5][5]
// Set all elements of A to 1.0.
A[:] = 1.0;
// Add elements 10->19 from A with elements 0->9 from B and place in elements 20->29 in C.
C[20:10] = A[10:10] + B[0:10];
// Element-wise equality test of B and C, resulting in an array of Boolean values, which are placed in A.
A[:] = B[:] == C[:];
```

Scheduler

- The Cilk Plus runtime uses a work-stealing scheduler to dynamically load-balance the tasks that are created by a Cilk Plus program;
- At a high level, the runtime scheduler executes a Cilk Plus program by using worker threads;
- In runtime, each worker thread maintains its deque using the following simple algorithm:
  - When a worker thread executes a cilk_spawn or a cilk_for statement, it may push new tasks onto the tail of its deque.
  - When a worker thread reaches a cilk_sync that needs to wait for a spawned function to complete, it tries to keep busy by popping a task from the tail of its deque.
  - If the worker thread discovers that its deque is empty, it tries to steal work from the head of the deque of another worker, where the worker is chosen at random.

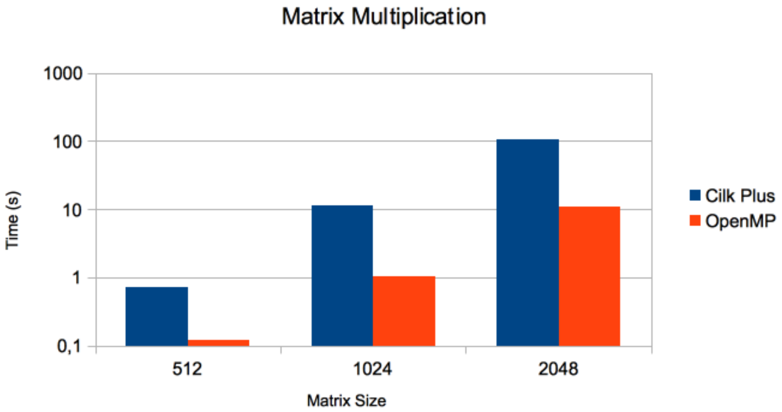

Figure : Results for matrix multiplication

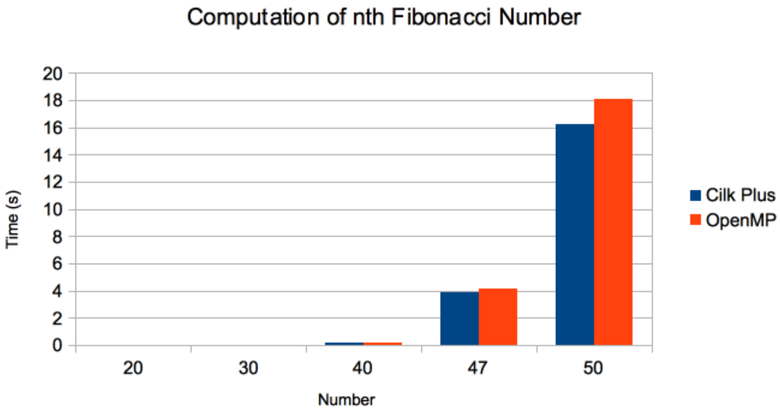

Figure : Results for fibonacci computation

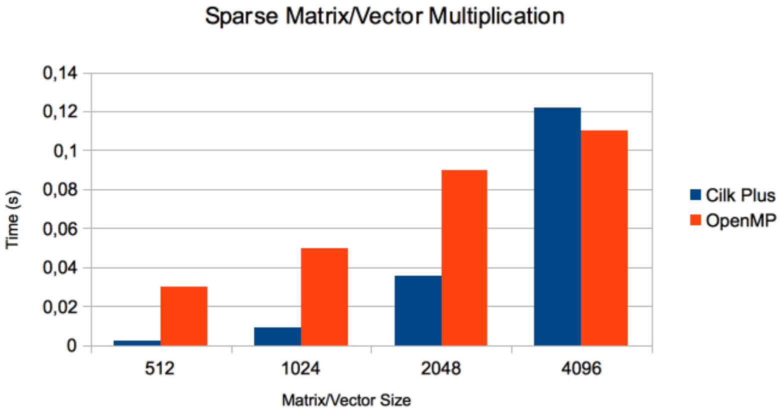

Figure : Results for sparse matrix/vector multiplication

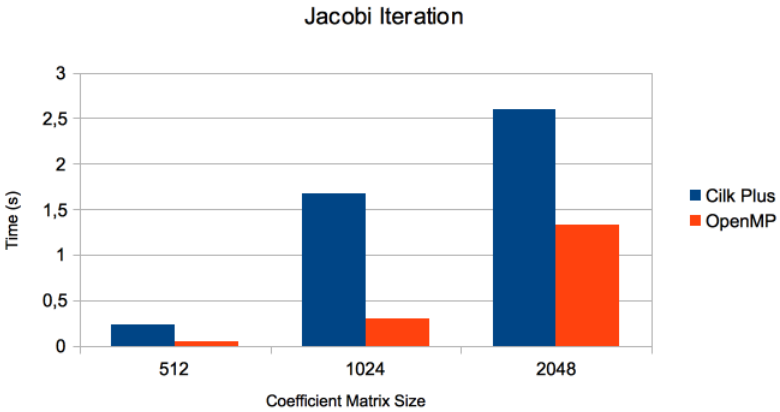

Figure : Results for jacobi iteration

Conclusions

- Ease of use makes it a powerful competitor;
- Not as powerful as OpenMP (although it can work together with TBB);
- Good for recursive algorithms;
- Spawning threads has very low overhead;
- Automatic load balancing is a plus;
- Supports incremental parallelization;

The Cilk Plus Extension

José Alves, Rui Brito

Universidade do Minho

Braga, May 2013

– ? –