Cache-oblivious wavefront algorithms for Dynamic Programming problems: probably efficient scheduling with optimal cache performance and high parallelism

Jesmin Jahan Tithi ¶ ¥, Pramod Ganapathi¶, Rezaul Chowdhury¶ and Yuan Tang*

¥Intel Corporation, ¶Dept. of Computer Science, Stony Brook University Stony Brook, New York, *School of Computer Science & Shanghai Key Laboratory of Intelligent Information Processing, Fudan University

What is Cache-oblivious Wavefront?

❑**Cache-oblivious Wavefront (Wave):** Variant of cache-oblivious recursive divide-and-conquer **(CORDAC)** algorithm with

- **Reduced or no artificial dependency** among subtasks
- Often with asymptotically **better parallelism**

❑**Cache-oblivious**: Do not use cache-parameter in algorithmic description **[unlike tiling/blocking]**

- **Cache oblivious ≠ cache-efficient**, but many CORDAC are cache-efficient

❑**Executes as if a wavefront is moving:** cells on grids are updated in a wavefront fashion

- **Examples:** stencil computations, **dynamic programming (DP)** algorithms

Benefits of Cache-oblivious Wavefront

Wave algorithms have order of magnitude better parallelism compared to the CORDAC algorithms

Algorithm	Longest common subsequence	Parenthesis/ Matrix Chain Multiplication	Floyd-Warshall's All pair shortest path
Wave	510	1916.0	1404
CORDAC	18	23	148

Table 1: *Parallelism in Wave and CORDAC algorithms estimated by the cilkview™ scalability analyzer on 16-core Sandy Bridge machine. Numbers show till how many cores the program should scale*

❑**Wave algos are more efficient on smaller grids, up to 2× faster than CORDAC on multicores, and up to (6×) on manycores**

Major contribution of this work

shows how to systematically transform CORDAC to COW

❑Keep structure similar to the CORDAC– gives cache optimality
❑Use analytically computed timing function to detect task readiness
❑**Can be scheduled using standard fork-join and a specialized hint-accepting scheduler**
❑No atomic-instructions/locks
❑Theoretically provable bounds

Our algorithmic approach eliminates shortcomings of prior cache-oblivious wavefront (PPoPP 2015) algorithms!

Example: in theory Wave has better span than CORDAC

LCS/Edit Distance	Span (lower better)	Cache complexity
Cache-oblivious wavefront	optimal $O(n \log n)$	optimal $O\left(\frac{n^2}{BM}\right)$
Recursive divide-and-conquer (CORDAC)	$O(n^{\log_2 3})$ optimal	$O\left(\frac{n^2}{BM}\right)$ (lower is better)

Table2: *Theoretical cache complexity and span of CORDAC vs Wave.* Here, B = block transfer size, M = cache size, n = input size,

Key Results

- **On Multicores (16-24 core Xeon):** Wave algorithms are around 2× faster than CORDAC
- **On Manycores (287 core Knights Landing/KNL):** Wave algorithms are around 4-6× faster than CORDAC

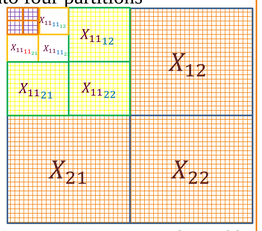
Recursive divide and conquer (CORDAC

❑ Divide the grid/matrix recursively into four partitions

❑ Solve each partition recursively respecting the **dependency among partitions (i.e., cells on grid)**

❑ Keep dividing until each partition becomes small enough

❑ Solve these small basecases iteratively



CORDAC on grid/DP table

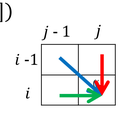
Example wavefront DP algorithm: Edit Distance

Edit Distance: Find minimum number of edits to convert string $S[1:m]$ to string $T[1:n]$ using Substitution, Delete, Insert edit operations.

DP recurrence and the dependency structure in the DP table:
Let, $ED[i][j]$ = cost of converting S of length i to T of length j
Then $ED[0][j] = j$ for $j = 0$ to n, $ED[i][0] = i$ for $i = 0$ to m
If i and $j \geq 1$,

$$ED[i][j] = \min \begin{cases} ED[i-1][j-1] + Sub(S[i], T[j]) \\ ED[i-1][j] + del([S[i]) \\ ED[i][j-1] + Ins(T[j]) \end{cases}$$

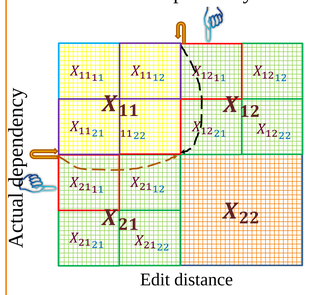
(i, j) depends on (i-1, j-1), (i, j-1) and (i-1, j)



Source of sub-optimal parallelism in CORDAC

❑**Artificial dependencies among tasks** at several granularities
❑**Artificial dependencies increase the span** (critical path length of DAG), and **reduce parallelism**

Actual dependency



Execution order of **edit distance**:
X_{11}
$X_{12} || X_{21}$ *(parallel)*
X_{22}

- Execution does not move to X_{12} or X_{21} until done with X_{11}
- But $X_{12_{11}}$, $X_{21_{11}}$ can start in parallel with $X_{11_{22}}$

Artificial dependency
Actual dependency

Cache-oblivious recursive wavefront technique removes artificial dependencies in CORDAC

By executing a task, as soon as all its actual dependencies are fulfilled

Cache-oblivious wavefront algorithms (formerly named "COW") were first proposed in PPoPP2015 [Tang, You, Kan, Tithi, Ganapathi, Chowdhury])

These COW algorithms **were complicated to develop, analyze, implement, and generalize**

Systematic way to transform CORDAC to COW

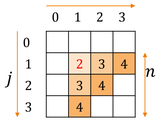
Step 1: Construct completion-time function

Completion time is the latest time when a cell gets updated/written

❑A closed-form formula that gives the timestep at which each DP grid cell is fully updated in wavefront order – an order in which cells are updated in the fastest wavefront algorithm

❑Max of (completion time of all input cells the cell depends on) + # input cells with that max time + 1

Example (Edit Distance):



❑Cell with index (i, j) lies on diagonal $(i + j)$

❑Cell (i, j) is fully updated at time step $(i + j)$

❑Completion-time, $C(i, j) = (i + j)$

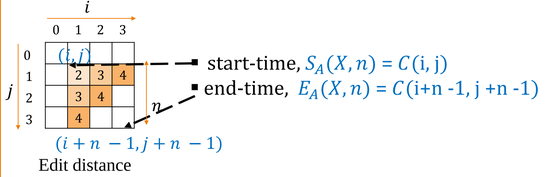
Step 2: Construct start- and end-time function for each recursive function type in CORDAC

❑ **Start time:** minimum (start time of all sub-functions called) + (wait time to avoid race, if any)

❑ **End time:** maximum (end time of all sub-functions called) + (wait time to avoid race, if any)

❑depend on the function type and the input and output parameters

Example: A region with top-left corner at (i, j) and dimension n



Step 3: Derive the recursive wavefront algorithm

❑ Augment each function in CORDAC algo. to accept a timestep parameter w

► spawn all functions in parallel provided for whom start-time $\leq w \leq$ end-time, remove all serialization in between

❑Each functions returns smallest timestep > w, for which it has some update to be applied -- used to find next value of w

❑Loop through all timesteps, w in non-decreasing wavefront order

Original CORDAC algo for Edit Distance

```
CORDAC(X, n) {
if(n<=switching_point) Iterative(X, n);
else {
      nn = n / 2;
      CORDAC (X11, nn);
      spawn CORDAC (X12, nn); CORDAC (X21, nn); sync;
      CORDAC (X22, nn);
  }
}
```

The transformed recursive wavefront code

```
CORDAC(X, n, w) {
if(n<=switching_point){
   if(S_A(X, n) = w) Iterative(X, n); return E_A(X, n);
}
else {
   nn = n /2;

   if(w<S_A(X11, nn))       w1 = S_A(X11, n)
   else if(w<=E_A(X11, nn)) w1 = spawn CORDAC(X11, nn, w);

   if(w<S_A(X12, nn))       w2 = S_A(X12, nn)
   else if(w<=E_A(X12, nn)) w2 = spawn CORDAC(X12, nn, w);

   if(w<S_A(X21, nn))       w3 = S_A(X21, nn)
   else if(w<=E_A(X21, nn)) w3 = spawn CORDAC(X21, nn, w);

   if(w<S_A(X22, nn))       w4 = S_A(X22, nn)
   else if(w<=E_A(X22, nn)) w4 = spawn CORDAC(X22, nn, w);

   sync;
   return min(w1, w2, w3, w4);//returns min w above the input w
   }
}
```

```
RecursiveWavefront(X, i, j, n) {
w = 0; max_completion_time = C(i+n-1, i+n-1);
while (w < max_completion_time)
      w = CORDAC (X, i, j, n, w);
}
```

Experimental Results

Generated algos for four dynamic programming problems

- Longest common subsequence (LCS) / Edit distance
- Parenthesis problem (Matrix chain multiplication)
- Floyd-Warshall's all pairs shortest paths (FW-APSP)
- Sequence alignment with general gap penalty (Gap problem)

Projected parallelism by CilkView™: x axis shows grid dimension and y axis shows till how many core the program should scale

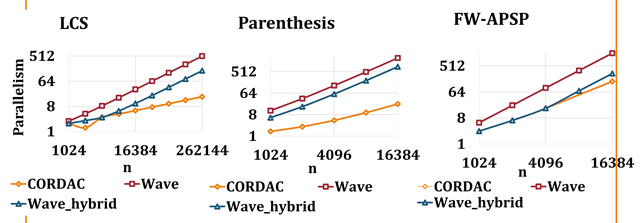


Figure: In this figure, Wave means recursive wavefront with pure iterative kernel after switching point, wave-hybrid means recursive wavefront with standard CORDAC algorithm after switching point

Speedup on 24-core Haswell and 71-core KNL compared to standard CORDAC

Algorithm	LCS	Parenthesis	FW APSP
wave	2×, 6×	2.6×, 4×	1.5×, 2×
wave-hybrid	2×	1.9×,	1.1x
COW (PPoPP'15)	1.9×	0.9	1.0

Table3: *Speedup achieved by wave wrt CORDAC on 24 core Haswell and KNL* (note: these code are neither vectorized, nor hand optimized)