




ARM  
2012 TechCon  
Join the community defining the future

SOFTWARE & SYSTEMS  
DESIGN

# Application parallelization for multi-core Android devices

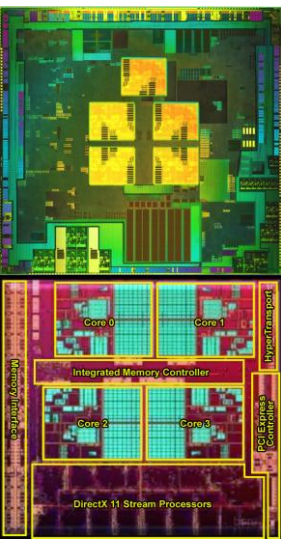
Jos van Eijndhoven  
Vector Fabrics BV  
The Netherlands  
<http://www.vectorfabrics.com>








ARM  
2012 TechCon  
Join the community defining the future

MULTI-CORE PROCESSORS:  
HERE TO STAY



- Multi-core processors are here to stay
- To make use of growing transistor count
- To allow run-time trade-offs between performance and power






ARM  
2012 TechCon  
Join the community defining the future

## GET MULTIPLE CORES TO WORK?

- 2 cores:  
Assume the OS provides **multiple processes** and/or kernel threads for workload
- 4 cores (and beyond):  
Requires **multi-threaded applications**
  - To obtain sufficient concurrent workload
  - To obtain top user experience

*Who makes such applications??*



ARM  
2012 TechCon  
Join the community defining the future

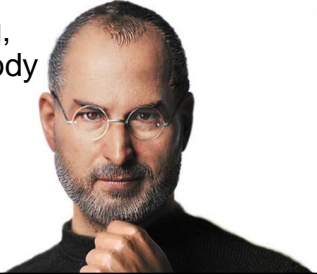
## PARALLEL PROGRAMMING IS HARDER THEN YOU THINK


**Herb Sutter, chair of the ISO C++ standards committee, Microsoft:**


“Everybody who learns concurrency thinks they understand it, ends up finding mysterious races they thought weren’t possible, and discovers that they didn’t actually understand it yet after all”

**Steve Jobs, Apple:**

“The way the processor industry is going, is to add more and more cores, but nobody knows how to program those things. I mean, two yeah; four not really; eight, forget it.”










ARM  
2012 TechCon  
Join the community defining the future

## PRESENTATION CONTENT

- Multi-threading concepts:
  - data- vs. task-partitioning
  - dependencies
- Concurrent programming
  - Patterns: reductions, functional pipelining
  - Pitfalls
  - Tooling support
- Android support
  - Use cases
  - Tooling
- Extension to GP-GPU
- Conclusion

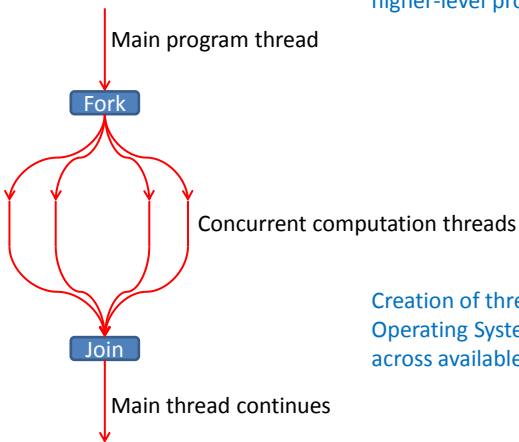





ARM  
2012 TechCon  
Join the community defining the future



## MULTI-THREADING: FORK-JOIN


Basic fork-join pattern, created through different higher-level programming constructs



Concurrent computation threads

Creation of threads is application responsibility. Operating System handles run-time scheduling across available processors!



## PARALLELIZATION : DATA- VERSUS TASK-PARTITIONING

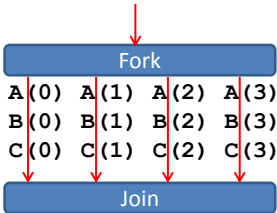
### Source code:

```
for (i=0; i<N; i++) {
    A(i);
    B(i);
    C(i);
}
```

### Sequential execution order:

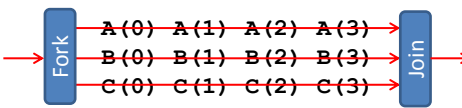
A(0)	A(1)	A(2)	A(3)
B(0)	B(1)	B(2)	B(3)
C(0)	C(1)	C(2)	C(3)



### Data partitioning:




(for large N, partition iterations over fewer threads)

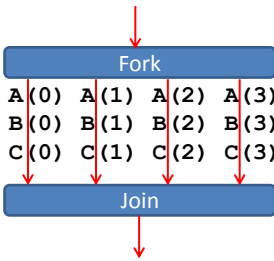
### Task partitioning:



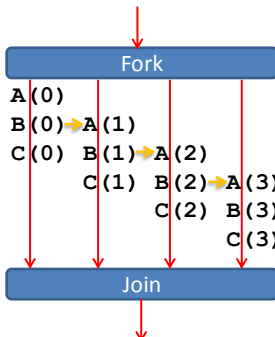





## ISSUE: DATA DEPENDENCIES





Maybe, B (i) produces a value that is used by A (i+1) ...




Adjust program source for parallelization:

- When feasible, remove inter-thread data dependencies
- Implement required data synchronization

Consciously choose task versus data partitioning, check dependency analysis!






## EXAMPLE: DATA DEPENDENCIES

**Variable assigned in loop body, used in later iteration**

```
// search linked-list for matching items
// save matches in 'found' array of pointers
for (p = head, n_found = 0; p; p = p->next)
    if (match_criterion(p))
        found[n_found++] = p;
```

Cannot (easily/trivially) spawn data-parallel tasks!

- No direct parallel access to list members **\*p**
- No direct way to assign index to matched item **n\_found**
- Maybe more problems hidden in **match\_criterion**








## EXAMPLE: ANTI DEPENDENCIES

**Storage location used in loop body, shared over iterations**

```
// convert table with floats to strings
char word[64];
for (i=0; i<N; i++)
{
    sprintf( word, "%g", table_float[i]);
    table_string[i] = strdup( word);
}
```

- Anti-dependencies are resolved by duplicating storage locations (thread-local storage)
- Need to make multiple copies of **word[]** space






## EXAMPLE: CONTROL DEPENDENCIES

**Control gives order constraints that hinder parallelization:**



```
// No creation of work beyond some point
for (i=0; i<N; i++)
{
    if (special_condition(i))
        break;
    // remainder of work is only OK after test
    table[i] = workload(i);
}
```


Since multiple threads proceed at non-determined speed (mutual order), above test risks violation in a data-parallel loop.



## PRESENTATION CONTENT

- Multi-threading concepts:
  - data- vs. task-partitioning
  - dependencies
- Concurrent programming
  - Patterns: reductions, functional pipelining
  - Pitfalls
  - Tooling support
- Android support
  - Use cases
  - Tooling
- Extension to GP-GPU
- Conclusion








## SOLVED: REDUCTION DATA DEPENDENCIES

- Reduction expressions: accumulate results of loop bodies with commutative operations:
 

```
// conditionally accumulate results
int acc = 0;
for (i=0; i<N; i++)
{
    int result = some_work(i);
    if (some condition(i))
        acc += result;
}
...use of acc ...
```
- Commutative operations are basic math as +, \*, &&, &, ||, but also more complex operations like 'add to set'.
- Parallelization can be achieved in (3?) different ways...



## SOLVED: REDUCTION DATA DEPENDENCIES



**Options to parallelize loops with reductions:**


1. Create thread-local copies of the accumulator. Accumulate over local copy in each thread. Merge the partial accumulators after thread-join. Eg. created automatically after:
 

```
#pragma OMP parallel for reduction(...)
```
2. Have one accumulator, synchronize updates through atomic operations. Eg. in C++11:
 

```
atomic_add_fetch( &acc, result);
```
3. Have one accumulator, synchronize updates through protection by acquiring and releasing semaphores. Eg. From the C++ Intel TBB:
 

```
concurrent_unordered_set
```



ARM  
2012 TechCon  
Join the community defining the future

## TASK PARALLELIZATION: STREAMING DEPENDENCIES

```


int A[N][M];

while (...)
{ produce_img();
  consume_img();
}

produce_img()
{ for (i ...)
  for (j ...)
    A[i][j] = ...
}

consume_img()
{ for (i ...)
  for (j ...)
    ... = A[i][j];
}

```



```

Thread1: while (...)
  produce_img();



Thread2: while (...)
  consume_img();


```

Synchronize thread progress:

- **True dependency:** consumer must wait for valid data
- **Anti dependency:** producer must wait with over-writing until after consumption



ARM  
2012 TechCon  
Join the community defining the future

## STREAMING DEPENDENCY: MODIFY DATA MODEL

```


int A[N][M];

while (...)
{ produce_img();
  consume_img();
}

produce_img()
{ for (i ...)
  for (j ...)
    A[i][j] = ...
}

consume_img()
{ for (i ...)
  for (j ...)
    ... = A[i][j];
}

```



```

concurrent_bounded_queue Aq;

Thread1: while (...)
  produce_img();

Thread2: while (...)
  consume_img();



produce_img()
{ for (i ...)
  for (j ...)
    Aq.push(...);
}

consume_img()
{ for (i ...)
  for (j ...)
    Aq.pop(...);
}


```

Channel access functions implement thread stall.





## CONCURRENT PROGRAMMING: MANY PITFALLS




**Introducing functional errors:**

- Overlooking use of shared/global variables (deep down inside called functions, or inside 3rd party library functions)
- Overlooking exceptions that are raised or caught outside studied scope
- Incorrect use of semaphores: flawed protection, deadlocks

**Unexpected performance issues:**

- Underestimation of time spent in added multi-threading or synchronization code and libraries
- Underestimation of other penalties in OS and HW (inter-core cache penalties, context switches, clock-frequency reductions)

**Parallel programming remains hard!**








## DEVELOPMENT OF PARALLEL CODE

**Guidelines:**

- Base upon a sequential program: functional and performance reference
- Apply higher-level parallelization patterns: clear semantics, re-use code, reduce risk
- Use tooling for analysis and verification
  - Prevent introduction of hard-to-find bugs
  - Prevent recoding effort that does not perform

**Managable development process!**









ARM  
2012 TechCon  
Join the community defining the future

## PRESENTATION CONTENT

- Multi-threading concepts:
  - data- vs. task-partitioning
  - dependencies
- Concurrent programming
  - Patterns: reductions, functional pipelining
  - Pitfalls
  - Tooling support
- Android support
  - Use cases
  - Tooling
- Extension to GP-GPU
- Conclusion




ARM  
2012 TechCon  
Join the community defining the future

## PAREON DATA PARTITIONING: SCHEDULE VIEW WITH SYNCHRONIZATION

**Insight in *proposed parallel solution*:**

- Data-partitioning with inter-thread data dependencies
- Speedup estimates based on anticipated schedule and overhead
- Clickable dependencies show properties and link to source code



Invocation	Speedup	Overhead	Streams
Loop_100806			
✓ Loop_101288	3.0	1 %	4

**ARM 2012 TechCon** **PAREON PREVIEW: TASK PARTITIONING ON PLAIN C CODE**

Project Profile Partitions My changes Log 2D-Profile Schedule Architecture PGrain.c Labs View Help Close

Data partitioning candidates

- Loop\_36507 cannot be subjected to data partitioning. There are 1 loop-carried memory clusters that you have chosen not to ignore: [Memory cluster 25].

Functional partitioning - Loop\_36507

Partition id	Threads	Speedup	Streams	Apply
Partition 1	4	2.6	3	Apply
Partition 2	4	2.6	2	Apply
Partition 3	3	2.4	2	Apply
Partition 4	3	2.2	2	Apply
Partition 5	2	1.9	0	Apply

Properties

Function: SELECT\_3\_p\_full\_cal

Line coverage: 30.8%

Uncovered lines

Invocation time

Estimated: 38.188 ns

Constraint: <click to edit>

Invocation statistics

Computation time: 28.812 ns (75.5%)

Memory penalty: 9.375 ns (24.5%)

DRAM traffic: 0 B

DRAM bandwidth: 0 B

Data cache statistics

Penalties

Level 1: 9.375 ns


Level 2: 0 ns

1.0 STEP-UP

**ARM 2012 TechCon** **VERIFICATION: PERFORMANCE MEASUREMENT**




For example: PERF 'flame graph'

- sampling-based profiling
- with view into kernel-level





## PAREON CODE INSTRUMENTATION AT COMPILATION

- Proprietary C/C++ compiler
- Creates instrumentation for run-time tracing of application activity (function entry/exit, loop entry/exit, ld/st addresses)
- Allows the analysis of ld/st address patterns in relation with loop nesting, across file scope.
- Allows code coverage analysis



## PAREON PARALLELIZATION PATTERN ANALYSIS

- Analysis detects interactions between loads & stores at different program locations to same memory address.
- Differentiate loop-inbound, loop-carried and loop-outbound dependencies
- Relate with malloc/free on same addresses



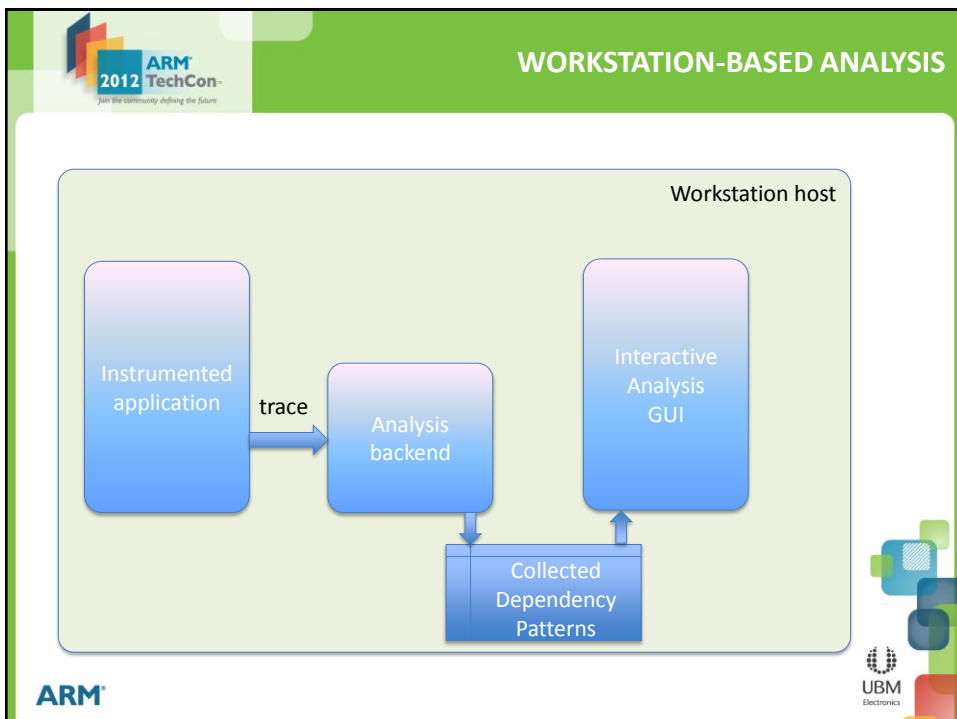
**ARM 2012 TechCon**  
Join the community defining the future

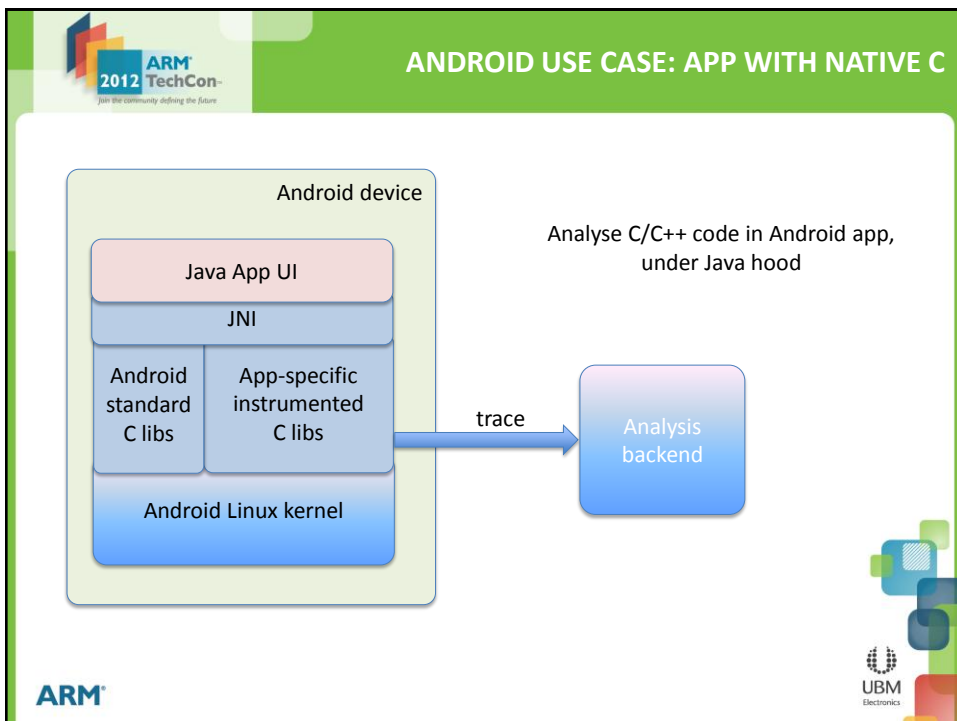
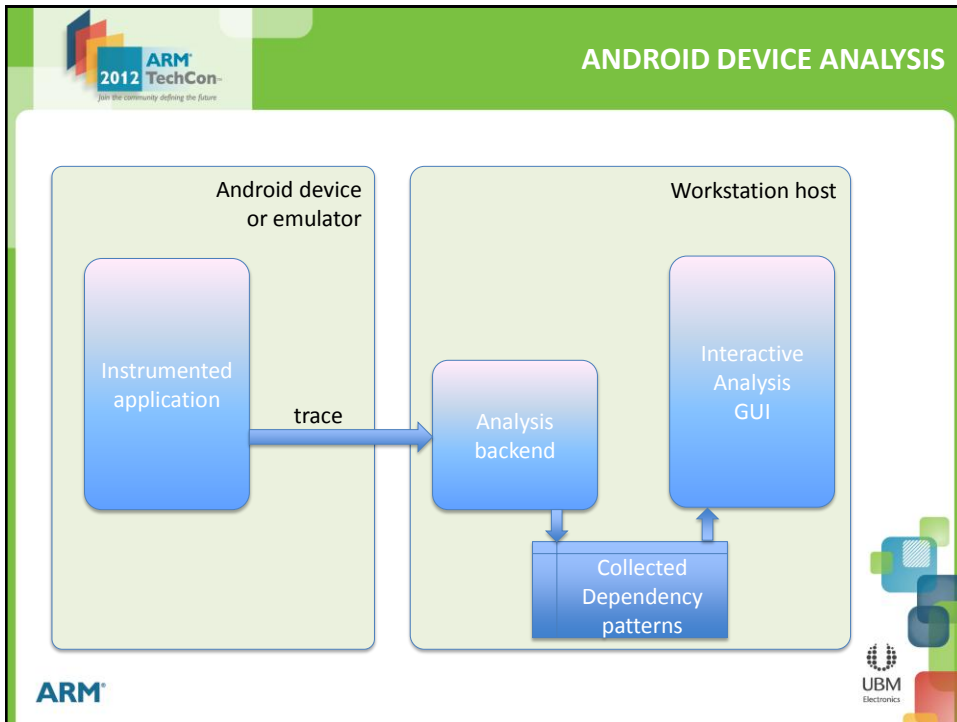
## PRESENTATION CONTENT

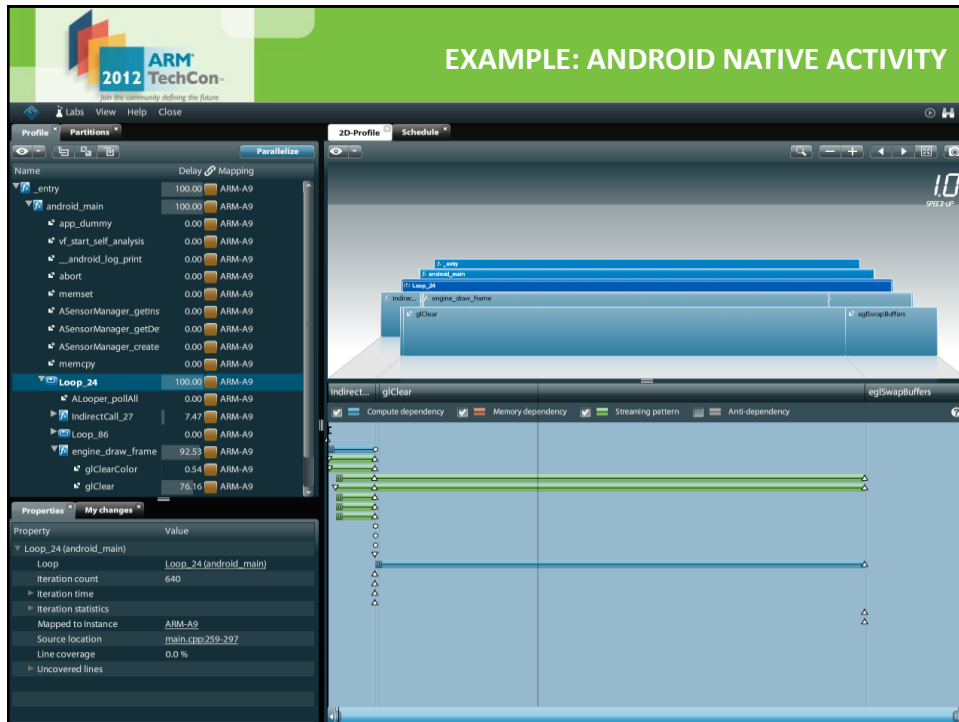
- Multi-threading concepts:
  - data- vs. task-partitioning
  - dependencies
- Concurrent programming
  - Patterns: reductions, functional pipelining
  - Pitfalls
  - Tooling support
- **Android support**
  - Use cases
  - Tooling
- Extension to GP-GPU
- Conclusion

**ARM**

UBM Electronics








**PRESENTATION CONTENT**

- Multi-threading concepts:
  - data- vs. task-partitioning
  - dependencies
- Concurrent programming
  - Patterns: reductions, functional pipelining
  - Pitfalls
  - Tooling support
- Android support
  - Use cases
  - Tooling
- Extension to GP-GPU
- Conclusion

ARM

UBM Electronics






ARM  
2012 TechCon  
Join the community defining the future

## FUTURE: CPU + GPU

Parallelization of C code not just across multi-core host CPU, also include GP-GPU mapping:

- Offload functionality (loop bodies) to GPU
- Verify CPU-to-GPU data dependencies
- Verify intra-GPU data dependencies
- Include performance model



Get more performance out of available silicon!




ARM  
2012 TechCon  
Join the community defining the future

## CONCLUSION

- There is a growing need for exploiting concurrency in applications
- Parallel programming remains hard:
  - Introduction of hard-to-locate bugs regarding dynamic data races and semaphore issues
  - Obtained speedup is lower than expected
- A sequential reference implementation helps to set a baseline.
- Proper tooling will save on edit-verify development cycles.









ARM  
2012 TechCon  
Join the community defining the future

**QUESTIONS**

- There is a growing need for exploiting concurrency in applications
- Parallel programming remains hard:
  - Introduction of hard-to-locate bugs regarding dynamic data races and semaphore issues
  - Obtained speedup is lower than expected
- A sequential reference implementation helps to set a baseline. ■
- Proper tooling will save on edit-verify development cycles.


ARM  
2012 TechCon  
Join the community defining the future

**Thank you for your attention**

Jos van Eijndhoven  
Oct 31, 2012

 **Vector** Fabrics