Survey Topics

(Embedded Systems - 2012-2013)

1. Memory subsystems for embedded computing: scratchpad, cache, and main memory for low-power, high-performance embedded systems.
2. Cache coherence for ring-based on-chip interconnects [G.Christoforakis]
3. Adaptive embedded computing (hardware, middleware, OS-level)
4. Light-weight cryptography for embedded computing

Example articles can be located via IEEE Xplore, ACM DL (Journal, Transaction articles and Conference papers: (MICRO, ISCA, CASES, ASPLOS, DATE, DAC))

(1)

Arslan Munir, Sanjay Ranka, and Ann Gordon-Ross, "High-Performance Energy-Efficient Multicore Embedded Computing", IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 23, NO. 4, APRIL 2012

Prasenjit Chakraborty, Preeti Ranjan Panda, "Integrating Software Caches with Scratch Pad Memory", CASES'12, October 7-12, 2012

(2)

Milo M. K. Martin, Mark D. Hill, Daniel J. Sorin, "Why On-Chip Cache Coherence is Here to Stay", Duke University Department of ECE Technical Report TR-2011-1

Michael R. Marty, "CACHE COHERENCE TECHNIQUES FOR MULTICORE PROCESSORS", Phd thesis, UNIVERSITY OF WISCONSIN – MADISON, 2008

(4)

Thomas Eisenbarth, Christof Paar and Axel Poschmann, Sandeep Kumar, Leif Uhsadel, "A Survey of Lightweight-

Cryptography Implementations", IEEE Design & Test of Computers, 2007

(3)

Examples of past and current conferences on adaptive computing

Grenoble, France -- October 11, 2009
http://www.artist-embedded.org/artist/APRES-09.html

Adaptive and reconfigurable systems can respond to environmental changes including hardware/software defects, resource changes, and non-continual feature usage, as well as provide support to live maintenance, extensibility and evolving requirements.

As such, these systems have the potential to exhibit an extended operations space and lifetime and improve efficiency in the use of system resources.

However, such flexibility also incurs overhead in terms of system complexity and resource requirements. For example, an adaptive system requires means for reconfiguration that allows it to adapt to changes. These means and their mechanisms introduce additional complexity to the design and the architecture, and they also require additional resources such as computation, power, and, for distributed reconfiguration, communication bandwidth.

Moreover, to take advantage of adaptability, new specification methods are needed, to define acceptable adaptation ranges which will be explored by the system at run-time to improve a given performance metric. However, current operating systems and network protocols are not designed to support such flexible requirements, and generally do not support complementary reflexive mechanisms that are needed to allow the application to adjust itself to the current configuration.

Finally, programming such systems also needs adequate middeware layers that provide adequate abstractions and interfaces for the development of adaptive applications. Building such middleware so that it preserves adaptability while providing performance guarantees together with satisfying other usual goals, such as modularity, reusability and scalability, is a challenge still to be conquered.

http://homepages.inf.ed.ac.uk/cdubach/adapt2013/

Embedded systems are rapidly evolving into heterogeneous machines featuring many processor cores. This leads to a tremendous complexity with an unprecedented number of available design and optimization choices for architectures, applications, compilers and run-time systems. Using outdated, non-adaptive technology results in an enormous waste of expensive computing resources and energy, while slowing down time to market.

http://www.see.ed.ac.uk/~ahs2013/CFP-AHS2013.pdf

Adaptation reflects the capability of a system to maintain or improve its performance in the context of internal or external changes, such as uncertainties and variations during fabrication, faults and degradations, modifications in the operational environment, incidental or intentional interference, different users and preferences, modifications of standards and requirements, trade-offs between performance and resources.

http://www.peccs.org/SANES.aspx

Self-adaptive systems are capable of monitoring their environment and their own states in order to optimize and match their functions to different scenarios and requirements. The concept of self-adaptiveness belongs to the wide scope of autonomic computing, while focusing on the aspects of self-awareness, context-awareness and self-adaptation. While networked embedded systems relate to several research domains (e.g. software engineering, hardware engineering, network and communication), their studies are usually limited in their own concerns, assumption and verification environment.

This special session aims to gather complementary research efforts addressing diverse issues in self-adaptive embedded systems, from requirement engineering, modeling, and application to concrete implementations.

Both functional and extra-functional (e.g. energy-efficiency, dependability and security) properties are of interest to the session.

Individual adaptive systems, distributed systems, and system-of-systems are relevant to this theme.

The session solicits works in the requirement engineering, software and hardware design, modeling, implementation and applications of self-aware, context-aware and adaptive networked embedded systems.

http://www.dcs.gla.ac.uk/conferences/lctes13/cfp.html

Embedded system design faces many challenges both with respect to functional requirements and nonfunctional requirements, many of which are conflicting. They are found in areas such as design and developer productivity, verification, validation, maintainability, and meeting performance goals and resource constraints. Novel design-time and run-time approaches are needed to meet the demand of emerging applications and to exploit new hardware paradigms, and in particular to scale up to multicores (including GPUs and FPGAs) and distributed systems built from multicores.

Embedded Processing

Architecture (HW, RTL/IP Core level) and Software (OS, C/C++)

1) Topic: Xilinx EDK

- Benchmark Kernels with threads on single core
- Benchmark Kernels on multiple cores (running at f1,f2,f3, etc)
- Benchmark Kernels with threads on cores with cache

2) Topic: Xilinx Vivado

Benchmark Kernels: hardware accelerators

3) Topic: Cache (design / benchmarking)

Cache:

2-way, 4-way set associative

4) Topic: Router/NI in Networks-on-Chip (design / benchmarking)

https://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/Router

5) Topic: Sorting & Searching N items (design)

6) Topic: Maximum-Likelihood Estimation (design / benchmarking)

Autoregressive–moving-average model
(http://en.wikipedia.org/wiki/Autoregressive%E2%80%93moving-average_model)

```
#define PROB_a 0.001 //0.001
#define PROB_b 0.001 //0.001

float Rt[1000] ;

float getSPRT(float M, float stddev, int current){
  float sum, SPRT ;
  int j;
     sum = 0 ;
     for(j=0; j<current; j++) {
        sum += Rt[j] - M/2 ;
     }
     SPRT = sum * M / stddev ;
     return SPRT ;
}

void wlpredict_thread(void)
{
      float limit_A, limit_B ;
      float stddev ;
      float M ;
      float avg, sum, SPRT, Tpred ;
      u32 timeptr, t0ptr ;
      int whole, thousands ;

      timeptr = 1 ;
      t0ptr = 0 ;
      limit_A = log( PROB_b / (1-PROB_a) ) ;
      limit_B = log( (1-PROB_b) / PROB_a ) ;
      stddev = 3;//2;
      M = 3 * sqrt(stddev) ;
      Rt[0] = 1;
      Rt[1] = 1;

      while ( 1 ) {
         timeptr++ ;

         Rt[timeptr-1] = 1;
         SPRT = getSPRT( M, stddev, timeptr ) ;

         while ( SPRT<limit_A || SPRT>limit_B) {
         u32 j ;

         // Update ARMA model:
              sum = 0 ;
              for(j=t0ptr; j<timeptr; j++) {
                 sum += Rt[j] ;
              }
```

```
            avg = sum / j ;
            stddev = fabs( (avg-3)*(avg-3) ) ;  // desired disturbance fixed to 3:
            M = 3 * sqrt(stddev) ;

            if (SPRT<limit_A) {
               Rt[timeptr-1] += fabs( Rt[timeptr-2] - (float) mb_iterations[0]/1000.0 )/4  ;
            } else if (SPRT>limit_B) {
               Rt[timeptr-1] -= fabs( Rt[timeptr-2] - (float) mb_iterations[0]/1000.0 )/4  ;
            }
            SPRT = getSPRT( M, stddev, timeptr ) ;

         }

         //Tpred = (float) local_temp + Rt[timeptr-1] ;
         Tpred = (float) mb_iterations[0]/1000.0 + Rt[timeptr-1] ;
      }

}
```

7) Topic: Smith–Waterman algorithm (design / benchmarking)