



*The Engine of SOC Design*

# Using configurable processors for high-efficiency multiple-processor systems

June 2006

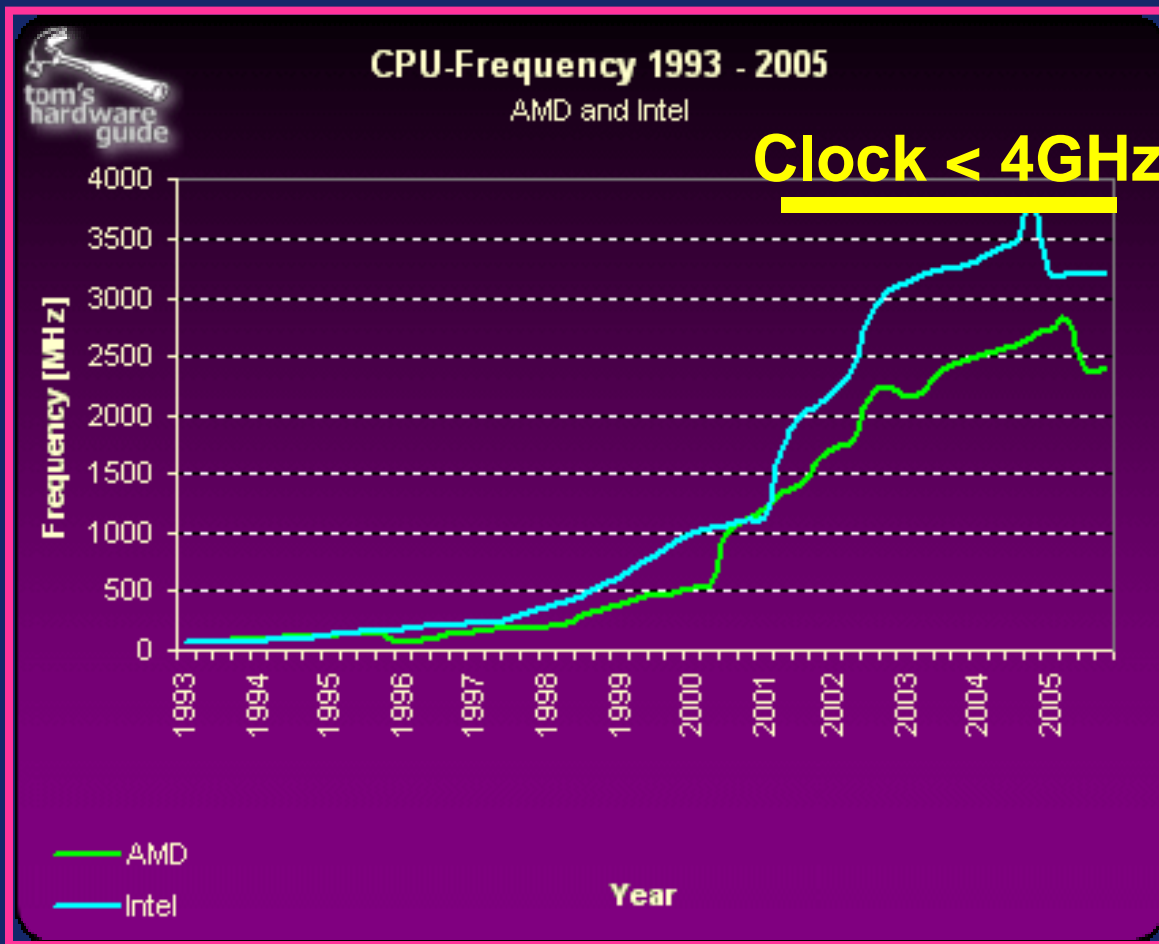
**Chris Rowen**

**President and CEO**

**Tensilica, Inc.**

# Why MP?

*Uniprocessors have hit the ceiling*



## Basic Implications:

1. Get performance from better architecture instead of more MHz
2. Use multiple processors



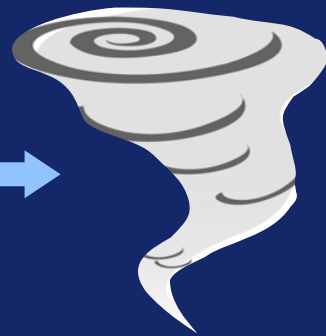
# Part I Processors to Order

Application-  
optimized processor  
implementation  
(RTL)

```
Stens Explorer GENERATED MAIN:
This XMP_core cannot be compiled in Stens Explorer. You must
it into the appropriate environment for host compilation.
Further, you should scan the file for two things. First, you
verify check to make sure that your system looks right. gcc
will in some cases not be able to generate a complete XMP_
such a case occurs you will see a comment noting that in the
below
*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "iso_xp.h"

static void loadProgram( XMP_core *cores, int nsaProc )
static int initCoresFromFile( FILE *fp, XMP_core *cores, XMP_
// number of processors
#define NUM_PROCESSORS 2
int XMP_main(int argc, char **argv)
{
    XMP_core cores[NUM_PROCESSORS];
    XMP_params params[NUM_PROCESSORS];
    XMP_multiAddressMapConnector router;
    XMP_memory *memories;

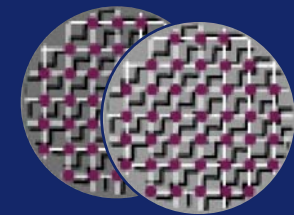
    unsigned int dontcare = 0x0; /* set addresses with aspEntr:
int i = 0;
while( i < argc )
```



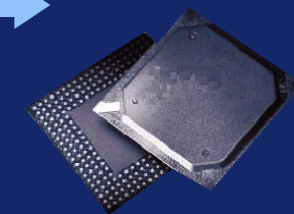
Tensilica  
Processor  
Generator

- Processor configuration
1. Select from menu
  2. Automatic instruction discovery (XPRES Compiler)
  3. Explicit instruction description (TIE)

Base CPU	OCD
Apps Datapaths	Cache
Extended Registers	FPU



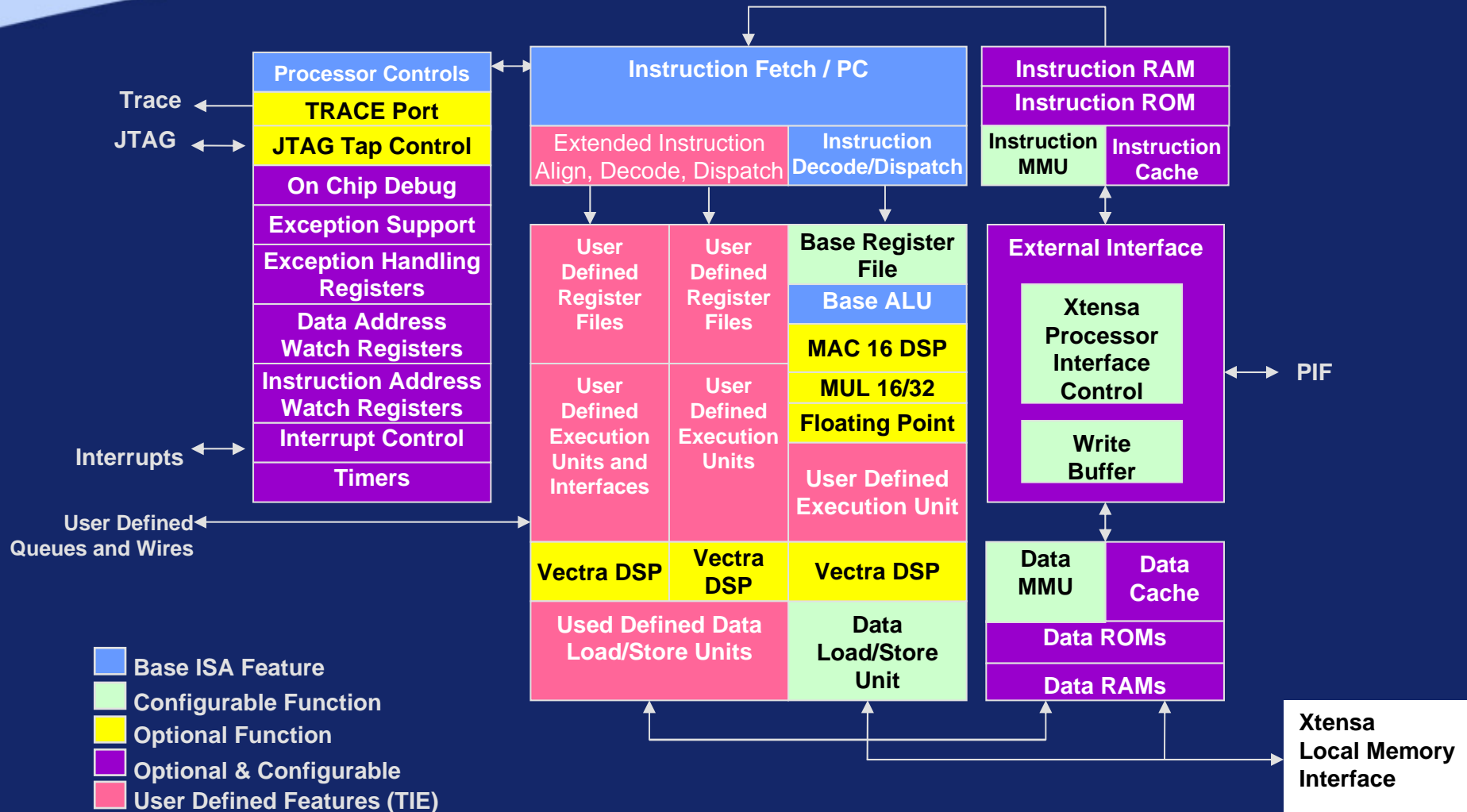
Tailored SW Tools:  
Compiler, debugger,  
simulators, OS ports



Build with  
any process  
in any fab

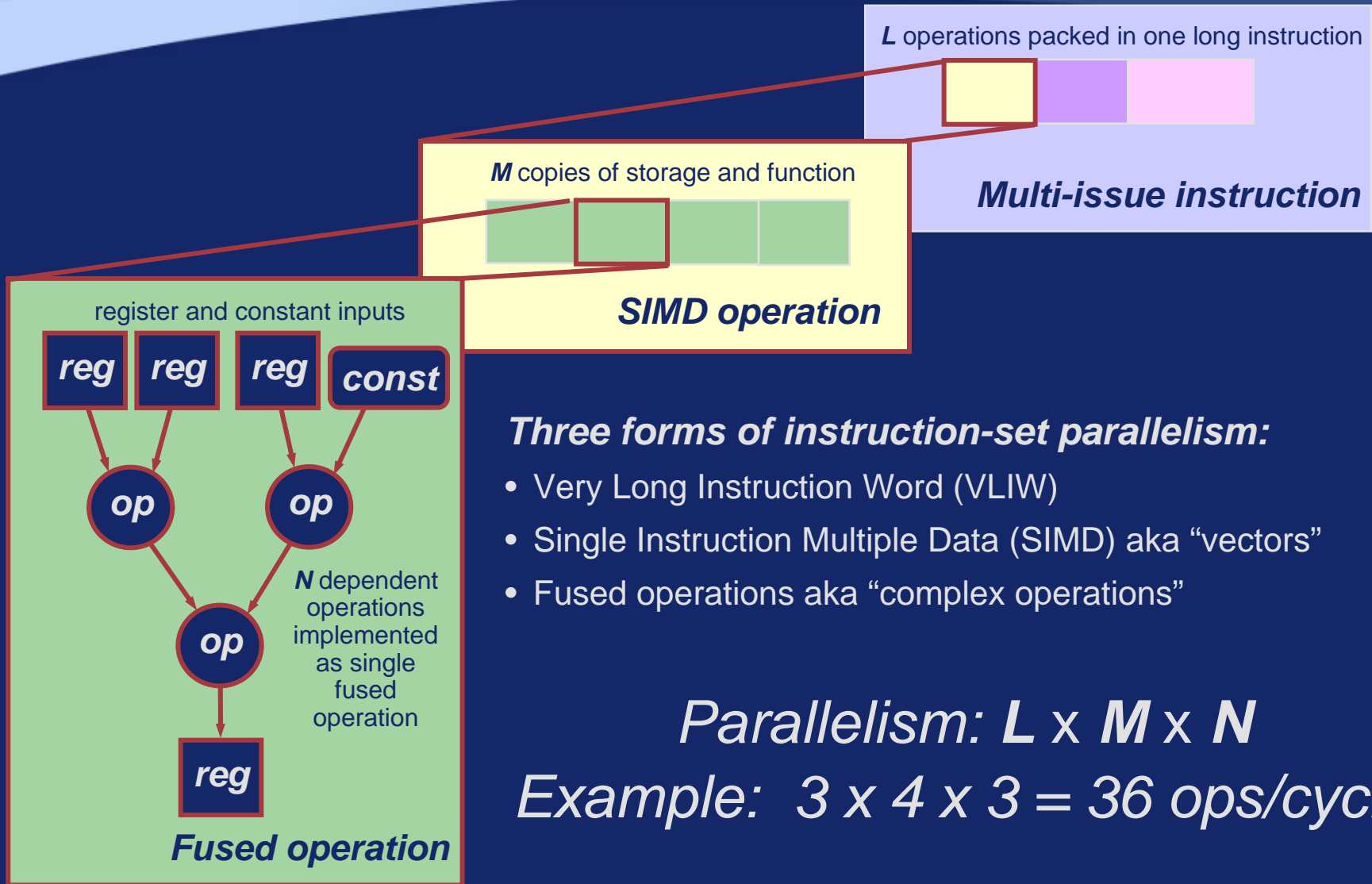


# Build Almost Any Processor



# Parallelism Inside the Processor

## Three Forms in Extensible Instruction Sets



### Three forms of instruction-set parallelism:

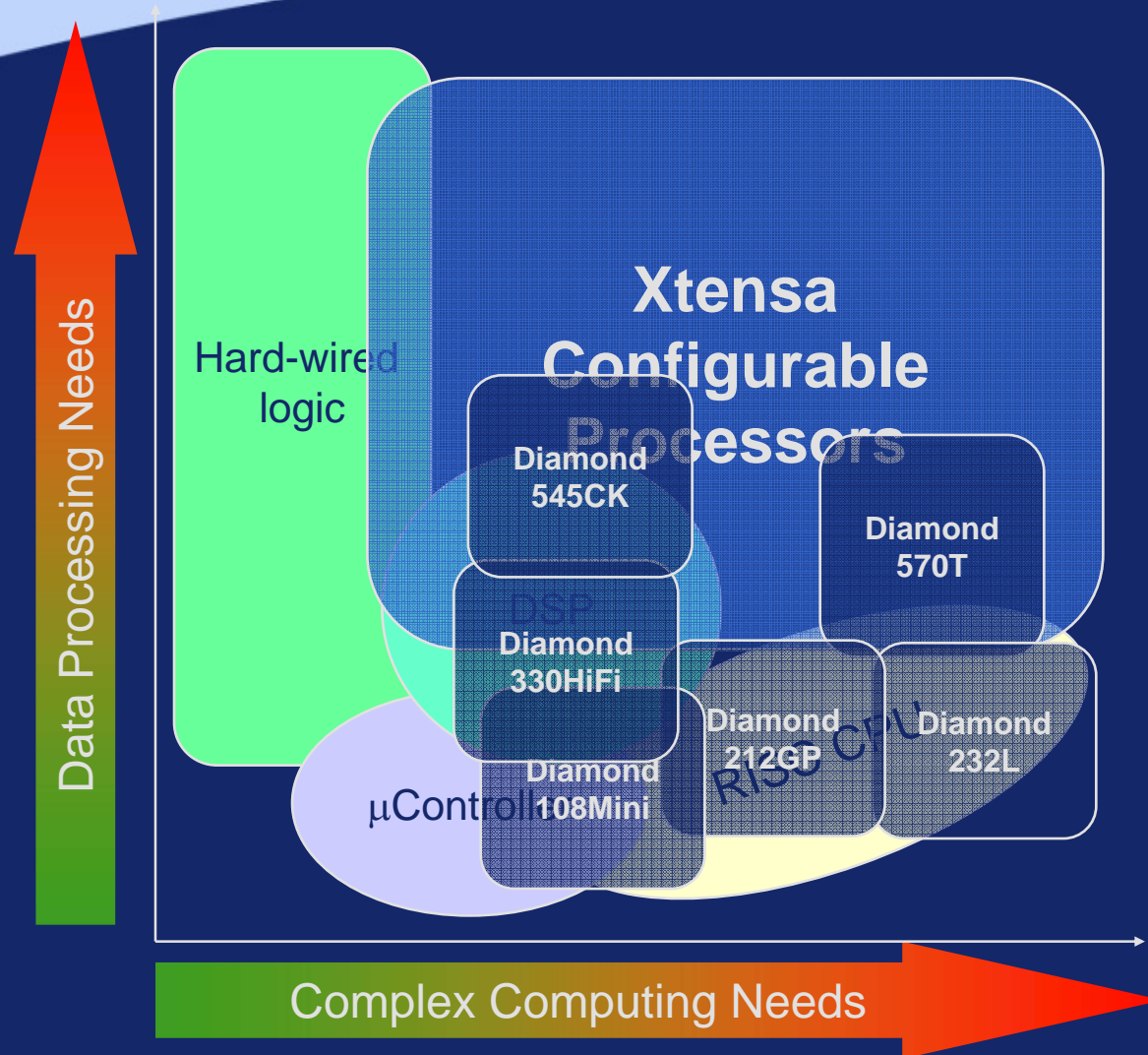
- Very Long Instruction Word (VLIW)
- Single Instruction Multiple Data (SIMD) aka “vectors”
- Fused operations aka “complex operations”

$$\text{Parallelism: } L \times M \times N$$

$$\text{Example: } 3 \times 4 \times 3 = 36 \text{ ops/cycle}$$



# Covering Breadth of SOC Demands



- Tensilica is the largest supplier of configurable processors
- Diamond Standard cores introduced February 20:
  - Broadest line of controller, CPU and DSP cores in industry
  - Highest performance synthesizable general-purpose CPU
  - Highest performance DSP
  - Most complete low-power audio solution
- Tensilica spans general-purpose and application-specific processors and software



# Automatic Instruction Set Optimization

General-purpose processors all look alike (more or less)

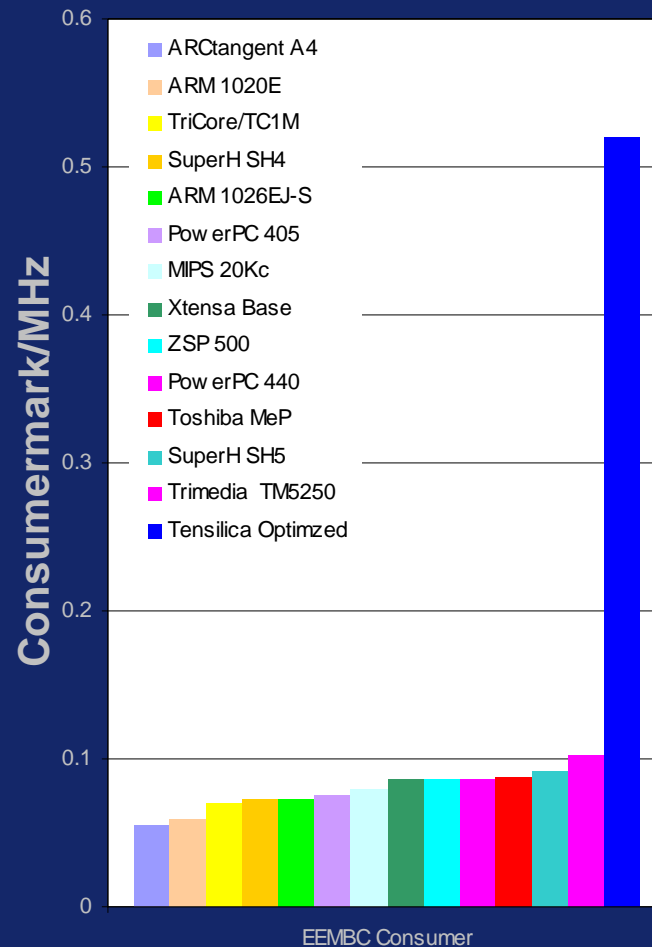
Automatic optimization from general-purpose C code with *XPRES Compiler*

- no intrinsic functions
- no assembly code
- no instruction set hardware definition coding

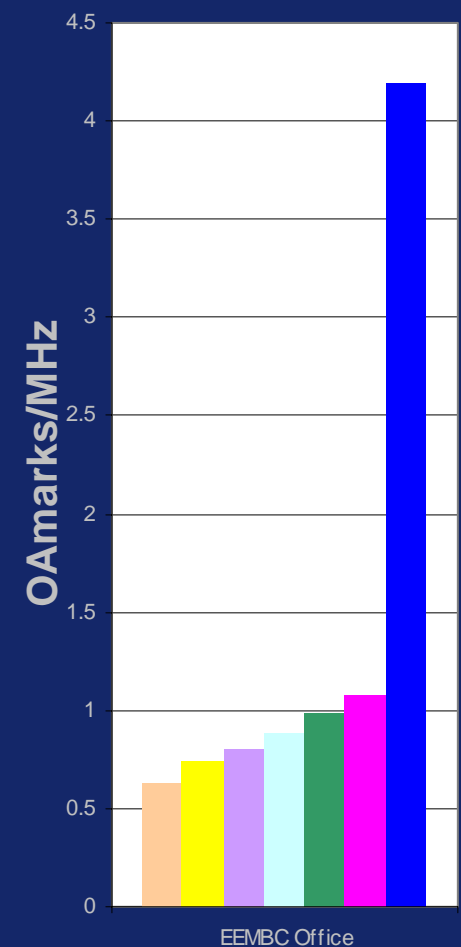
Results for all reported cores:

- Instruction set synthesis beats all other processors by 4-8x

Consumer Electronics



Office Automation

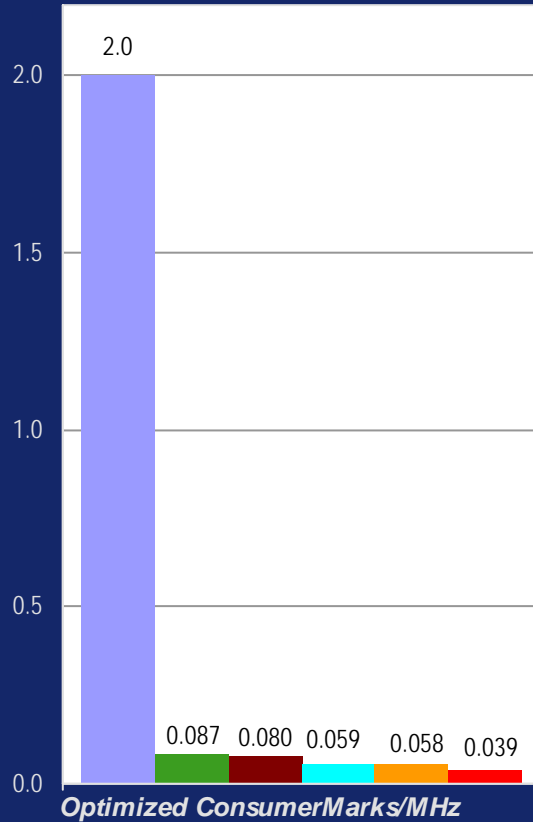




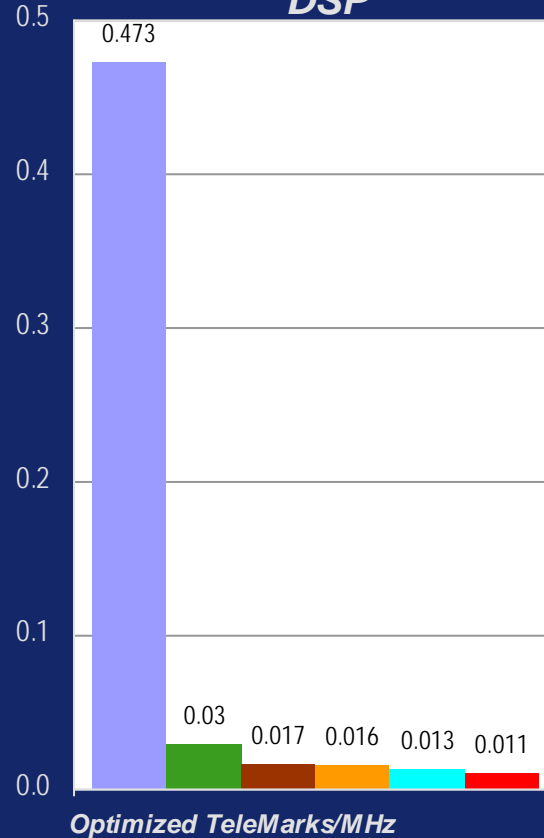
# Explicit Instruction Set Optimization

## Tensilica Instruction Extension

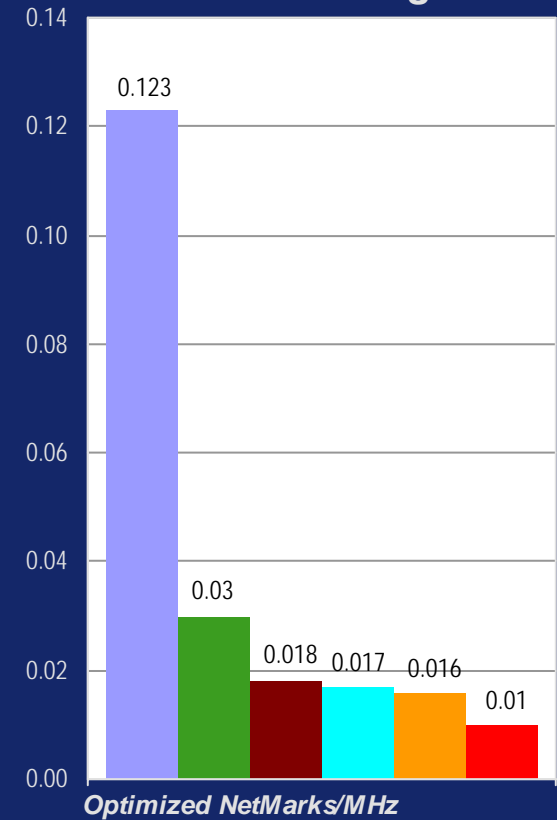
### Consumer Electronics



### DSP



### Networking



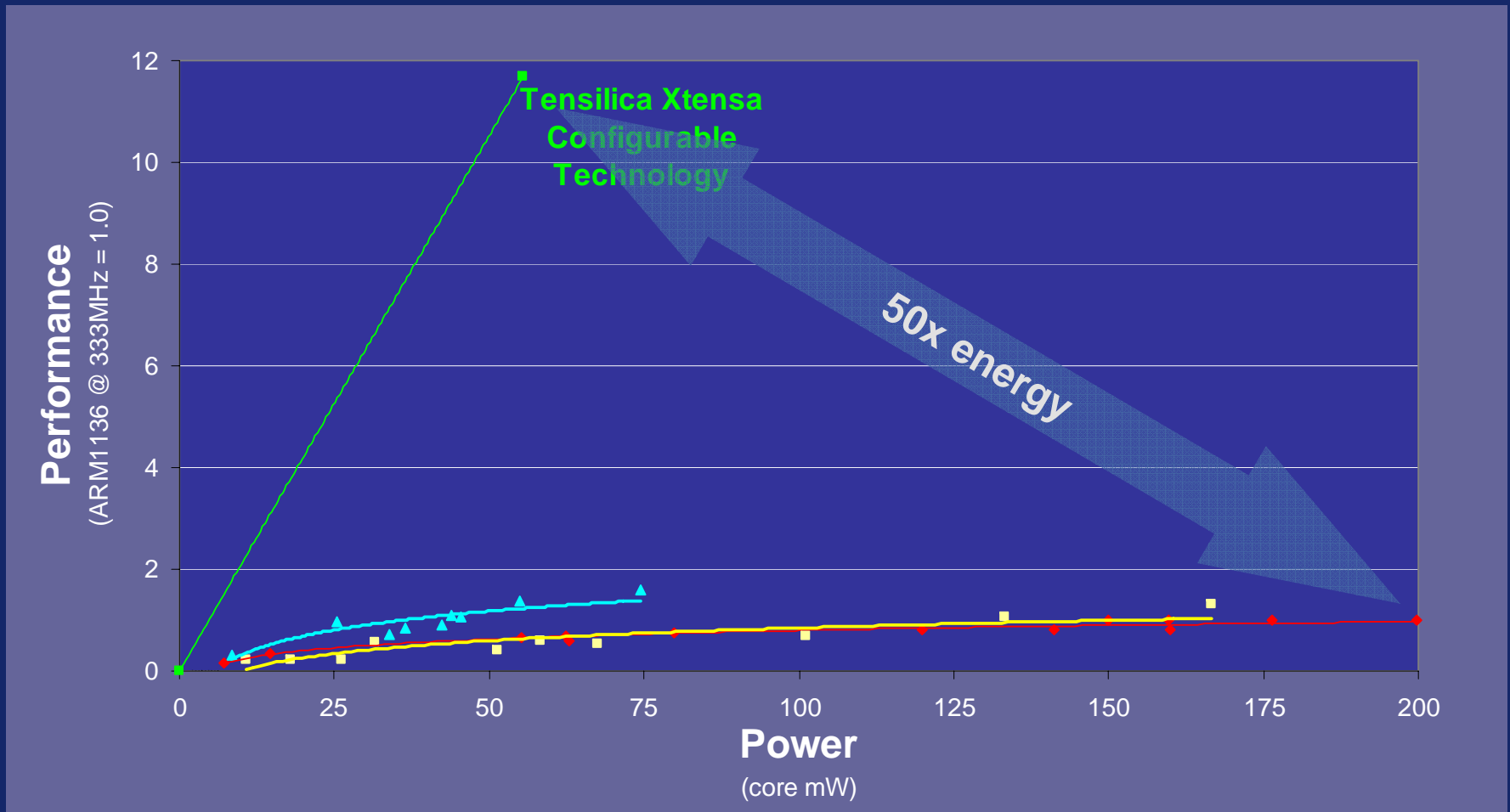
Optimized NetMarks/MHz

- Extensible optimized
- Extensible out-of-box
- MIPS64 20Kc
- ARM1020E
- MIPS64b (NEC VR5000)
- MIPS32b (NEC VR4122)



# Processor Power and Performance

## Xtensa Application-Specific Cores



Performance on EEMBC benchmarks aggregate for Consumer, Telecom, Office, Network, based on ARM1136J-S (Freescale i.MX31), ARM1026EJ-S, Tensilica Diamond 570T, T1050 and T1030, MIPS 20K, NECVR5000). MIPS M4K, MIPS 4Ke, MIPS 4Ks, MIPS 24K, ARM 968E-S, ARM 966E-S, ARM926EJ-S, ARM7TDMI-S scaled by ratio of Dhrystone MIPS within architecture family. All power figures from vendor websites, 2/23/2006

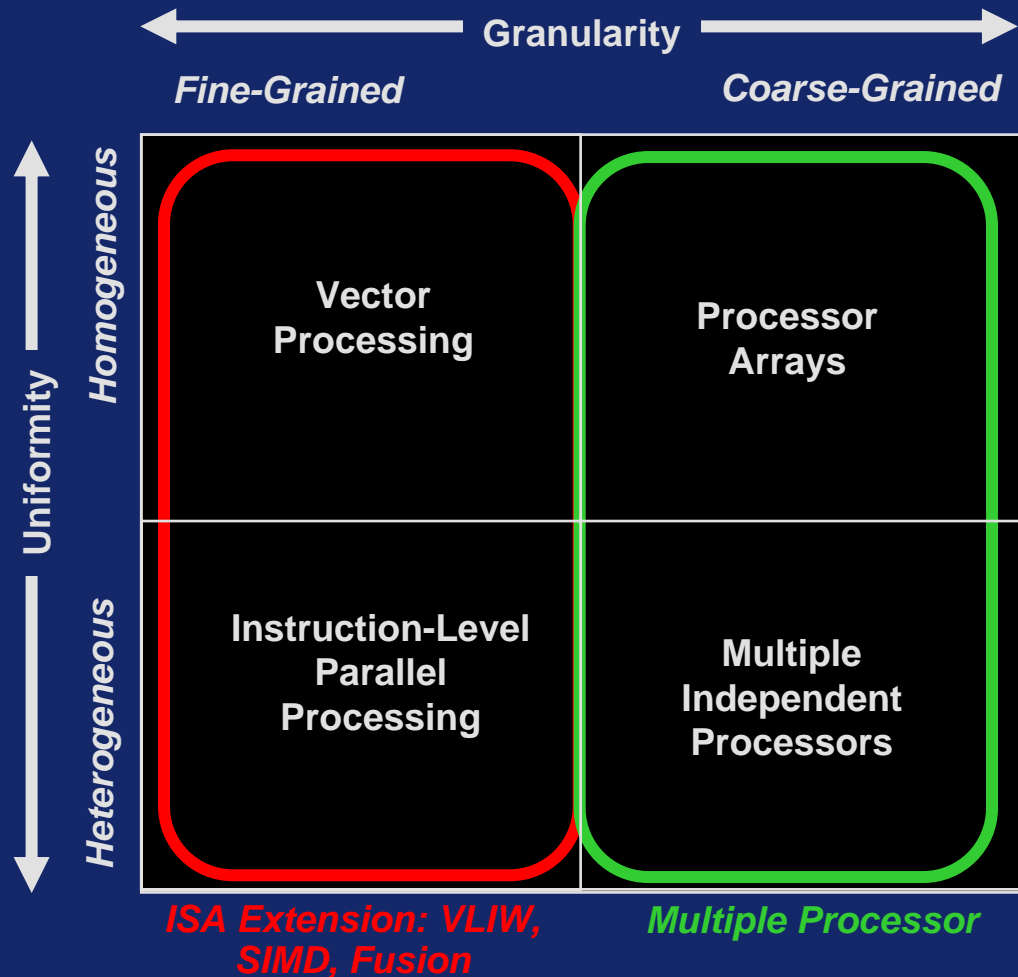


# Part 2 Multiple Processors

- Moore's Law scaling allows many more functions per chip
- Most applications have multiple forms of available concurrency
- Multiple concurrent processors much lower energy than a big processor

### Examples:

- Dual core x86
- IBM/Sony Cell 1+8 Processor
- Cisco CRS-1 188 SPP





# Automated MP SOC Design Flow

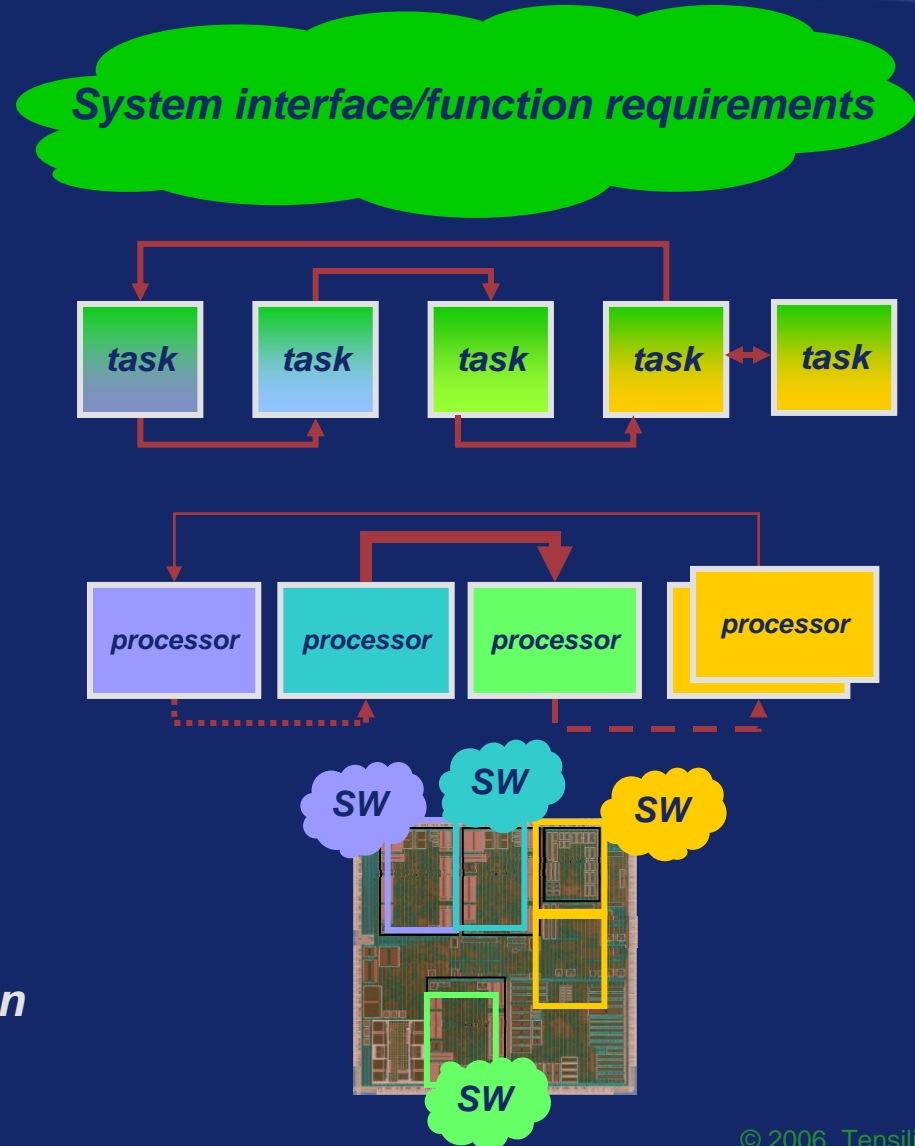
*Application → Tasks → Processors → Final HW+SW*

*Complex system definition*

*Rapid partitioning into a set of communicating tasks*

*Optimized HW/SW communication on tuned processors and interconnect*

*Physical synthesis of hardware + high-accuracy system software integration*





# Low Power and Multiple Processors

Low power and high performance

## Why MP?

- Real systems implement multiple concurrent functions
- Demand for flexibility turns multiple hard-wired blocks into multiple processors

## MP from the ground up

- MP architecture
- MP SW development tools
- High bandwidth/low latency
- Novel MP interconnects



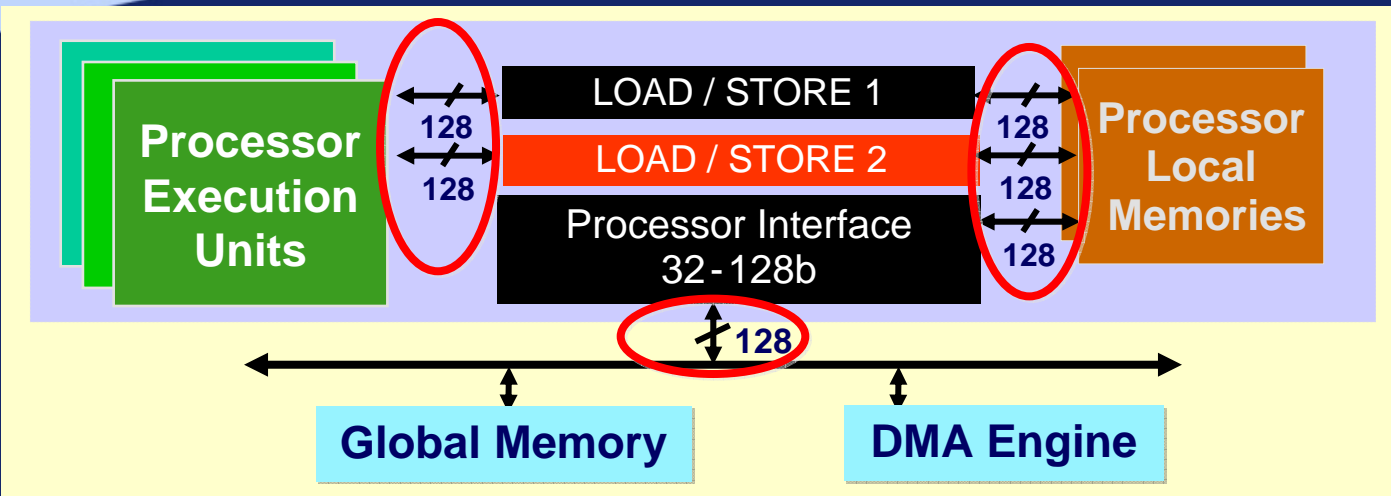
Tensilica Diamond 570T  
~0.8mm<sup>2</sup>  
~90mW @ 525MHz



3 x minimum Xtensa 6  
~0.36mm<sup>2</sup> total  
~50mW @ 525MHz

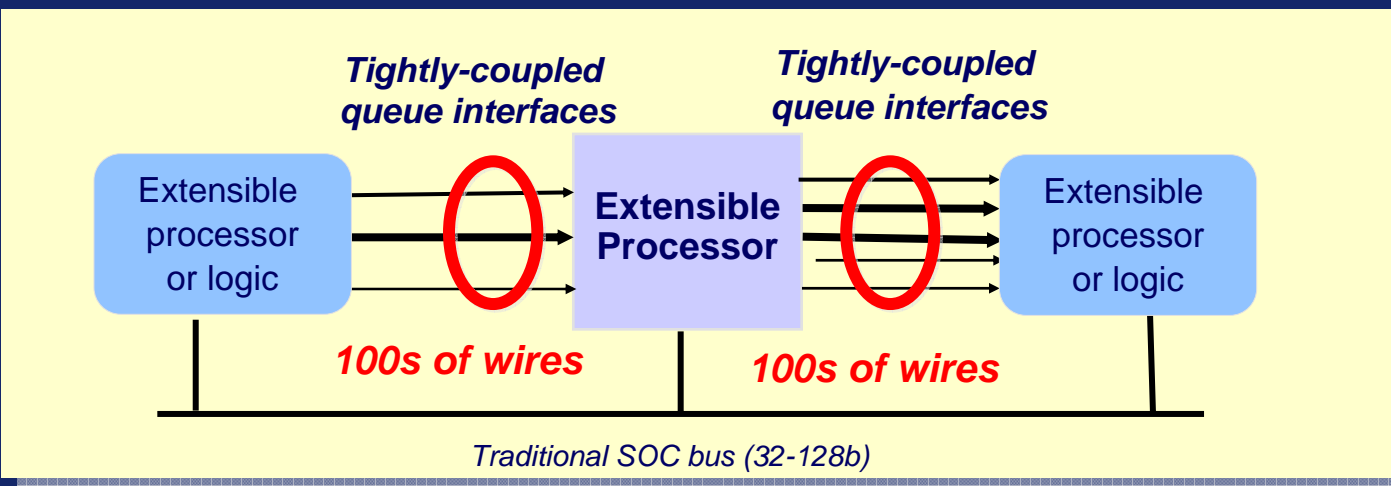
# Keys to Efficient MP

*High bandwidth, low energy*



## Remove RISC bottlenecks

- > 23 GB/s memory bandwidth @500 MHz
- Energy-efficiency for memory-based tasks



## Remove bus bottlenecks

- Arbitrary size queues
- Zero - overhead synchronization
- Lowest energy communication



# Optimizing Data Movement

## Traditional processor-processor data transfer:

1. Proc 1: application writes local mem
2. Proc 1: OS reads local mem, writes shared mem
3. Proc 2: OS reads shared mem, writes local mem
4. Proc 2: application reads local mem

**Result: 2 bus transfers, 3 reads, 3 writes**

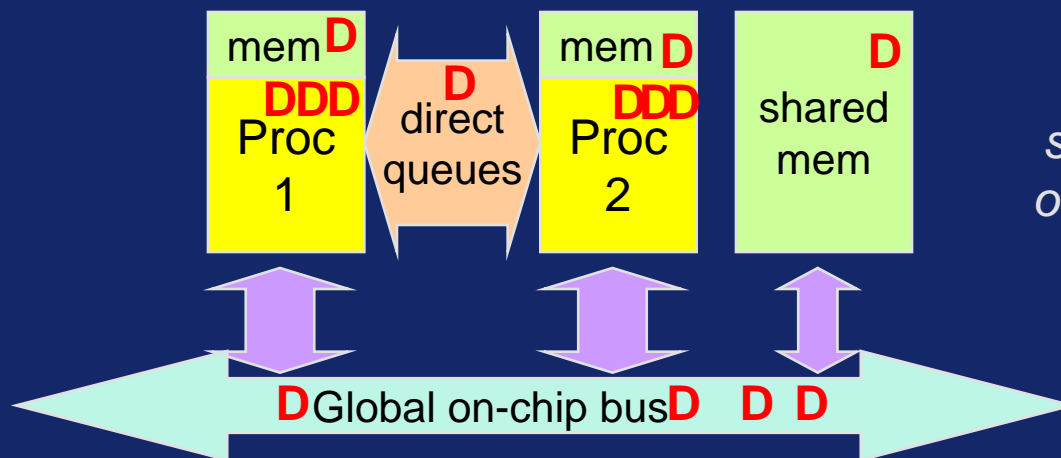
**Optimized: 2 bus transfers, 1 read, 1 write**

## Optimized processor-processor data transfer:

1. Proc 1: application writes to Proc 2 mem
2. Proc 2: application reads local mem

**Result: 1 bus transfer, 1 read, 1 write**

**Optimized with direct queues: 0 bus transfers, 0 reads, 0 writes**



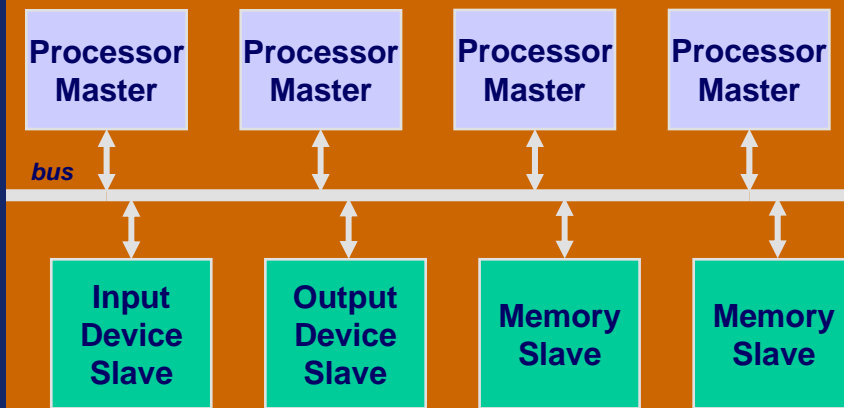
*Memory and bus structures consume lots of power when accessed*



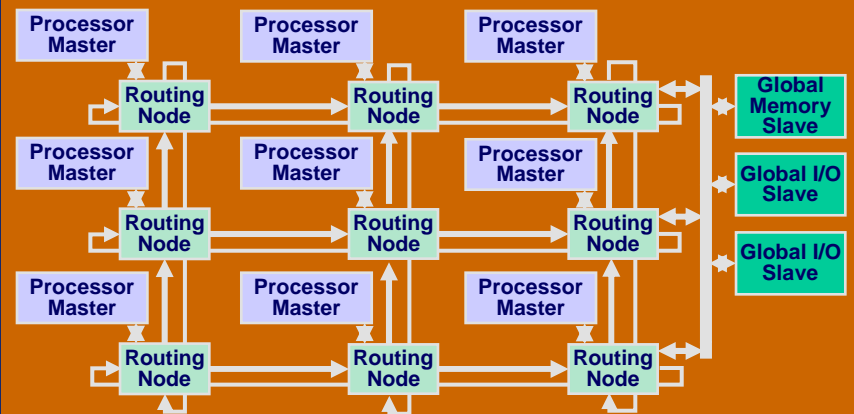
# Keys to Efficient MP

*Flexible range of topologies*

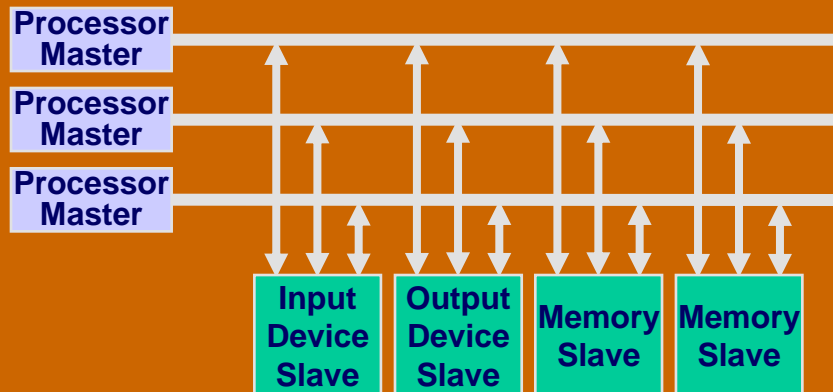
## Shared Bus



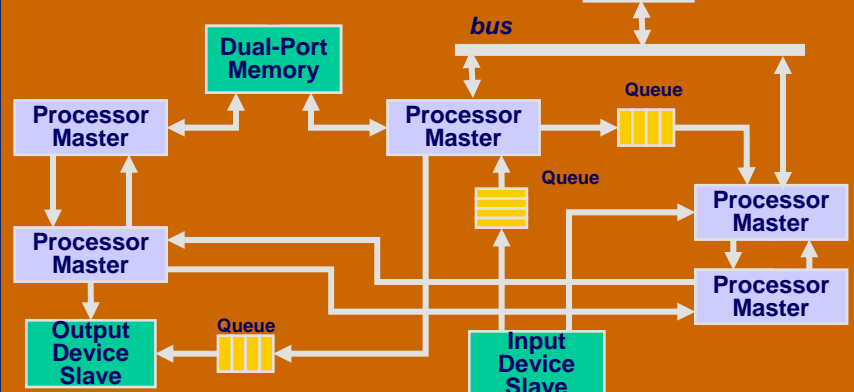
## On-chip Routing Network



## Cross-Bar



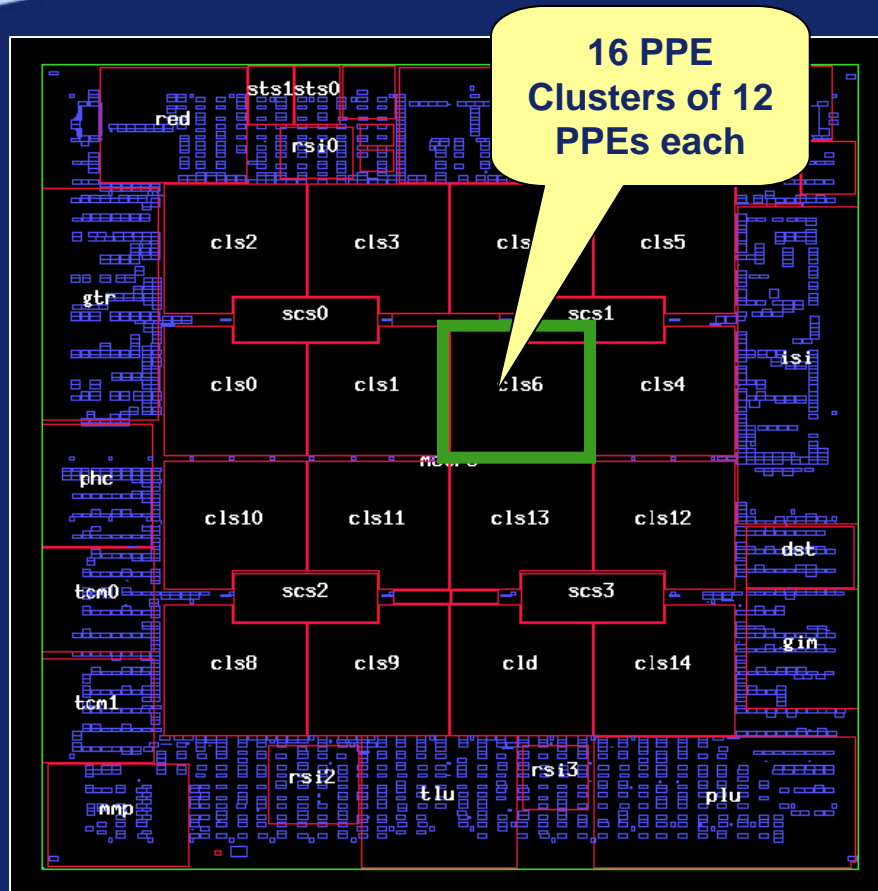
## Application-specific





# Enabling Parallel Architectures

## Cisco CRS-1 Terabit Router



188 Xtensa network processing cores per  
Silicon Packet Processor  
Up to 400,000 processors per system



# What About MP Programming Models?

Despite 20-30+ years of research, there is no generally accepted, 'good' method to automatically find or infer concurrency from arbitrary application code

There are many programming models for SMT, SMP, AMP – often provided as add-on libraries

- Eg. OpenMP, MPI (message passing interface), etc.
- Users manually modify code to identify
  - Possible concurrent tasks or threads
  - Communications mechanisms
  - Synchronization mechanisms

Very coarse-grained (whole application) concurrency – data-wise concurrency – does not need to modify source code, but needs an MP scheduler of some sort

Best, and reasonable, option today is the use of MP API's for communications and synchronization

Three common religious sects:

- **Message passing**
- **Shared memory**
- **SHMEM (explicit data block sharing)**



# Requirements for MP System Design

## Integrated Development Environment (IDE)

- Define applications
- Configure processors and extend them
- Profile single applications on single processors
- Develop MP structure
- Map tasks to processors
- Generate simulations, launch, trace, analyse, iterate

## Standards for MP structural definition

- Eg. XML as in SPIRIT

## Standards for IP model interoperability

- SystemC, TLM, beyond OSCI TLM

## Both cycle accurate and fast simulators

- System analysis and verification and SW validation

## Abstract programming model(s)

- Pipelined dataflow
- Multi-threaded
- More easily re-map tasks to processors



# Eclipse-based Xtensa Xplorer Design Cockpit Implements and Automates MPSOC Flow

## Software Development Environment

- C code development
- Graphical debugging
- C project management
- Code profiling, tuning
- Pipeline-accurate view

## Processor Optimization Environment

- TIE code development for extensions
- Configuration option management
- Instant gate count

## SOC System Architecture Exploration

- Multiple processor system simulation
- Multiple core debug
- Interactive model generation
- SLD

The screenshot displays the Eclipse IDE interface for the Xtensa Xplorer Design Cockpit. It features three main windows:

- Static Pipeline Preview:** Shows a table of instructions and their pipeline stages (I, R, E, M, W) across 11 cycles. The instructions include: `l32i.n a4, a1, 16`, `movi.n a5, 1`, `s32i.n a5, a1, 24`, `bge a3, a4, 40000bd7 <ma...`, `movi.n a9, 0`, `s32i a9, a1, 24`, `l32i a10, a1, 24`, `beqz.n a10, 40000c16 <m...`, `ld_vr v2, a1, 8`, `l32i.n a3, a1, 16`, `l32r a4, 40000868 <.Debu...`, and `addx8 a3, a3, a4`.
- C/C++ Development:** Shows the source code for a `MULADD16` instruction. The code includes: `wire [31:0] mtmp1;`, `wire [31:0] mtmp2;`, `assign mtmp1 = oper0[15:0] * oper1[15:0];`, `assign mtmp2 = oper0[47:32] * oper1[47:32];`, and `assign oper2 = {oper2[63:32] + mtmp2, oper2[31:0] + mtmp1};`
- Hardware Configuration:** A table showing memory and peripheral configurations for different components.

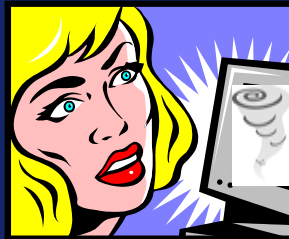
	sram(256M)	srom(4M)	dataram(8K)	instram(8K)	dram2(8K)
	0x20000000 0x203fffff				
	0x40000000 0x4fffffff		0x51f00000 0x51f1ffff		
				0x51f00000 0x51f1ffff	
					0x51f00000 0x51f1ffff
	0x20000000 0x203fffff				
	0x40000000 0x4fffffff				
					0x51f00000 0x51f1ffff



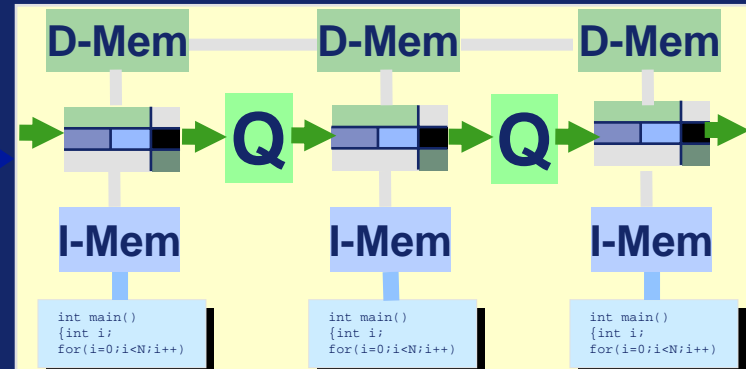
# Application → Platform Flow

```
int main()
{
  int i;
  short c[100];
  for (i=0;i<N;i++)
  {
```

Original Version of  
Algorithm/Application  
Source Code



Design  
Flow and Tools



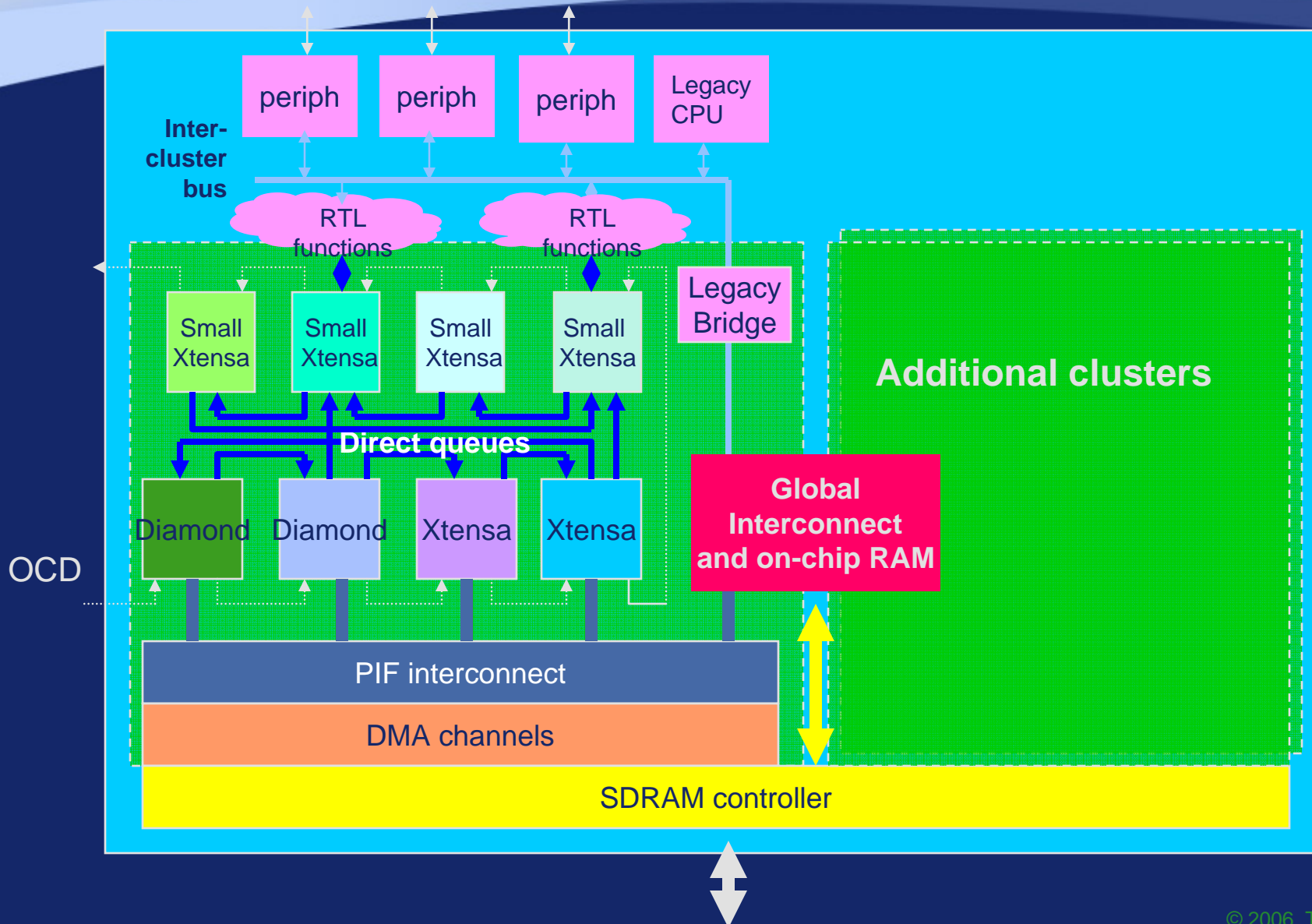
Configured  
Multiprocessor  
Subsystem

## SLD Capabilities support:

- Partition algorithmic/application code into tasks
- Define inter-task communication with rich abstract API
- Generate optimized task engines for specific tasks
- Map communications to queues and memories
- Simulate and explore design space
- Generate optimized, configured, multiprocessor subsystem



# System Structure: Hierarchy of Clusters

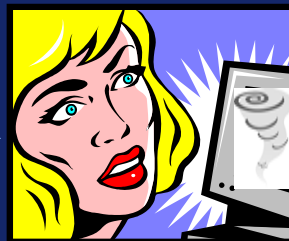


# Different Multi-processor Design Flows

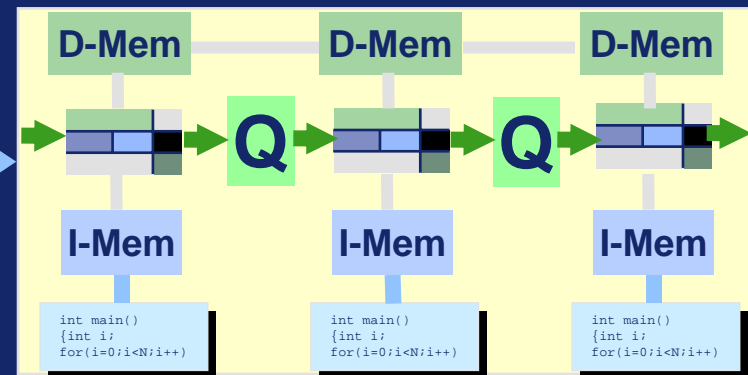
## Top-Down

```
int main()
{
  int i;
  short c[100];
  for (i=0;i<N;i++)
  {
```

Original Version of  
Algorithm/Application  
Source Code



Design  
Flow and Tools

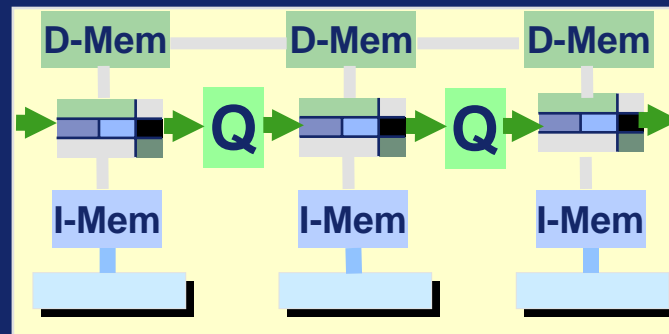


Configured  
Multiprocessor  
Subsystem

## Middle-Out (Platform-Based)



HW-centric  
22 Architect



Define Platform



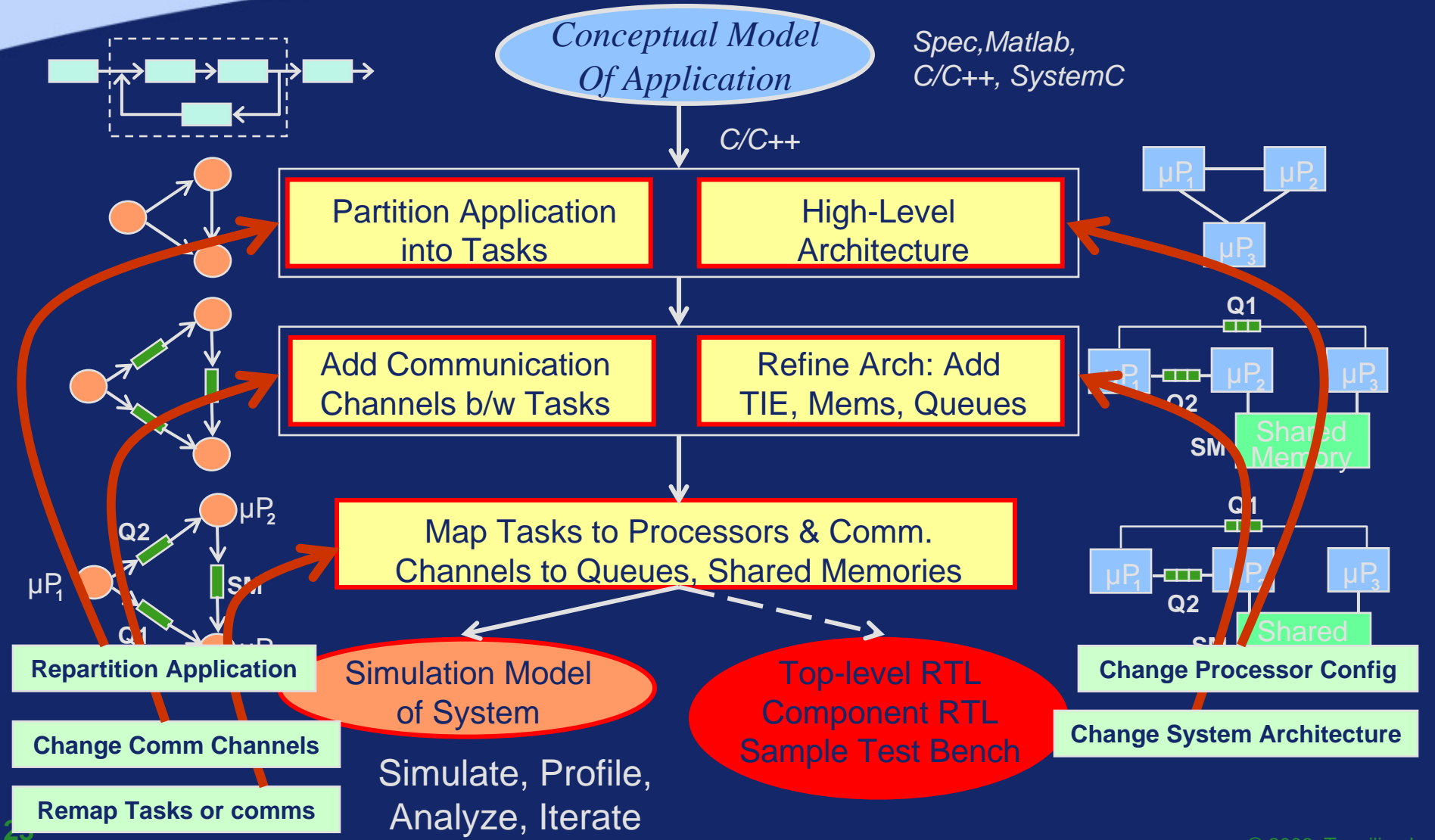
Applications SW  
Specialist

```
int main()
{
  int i;
  short c[100];
  for (i=0;i<N;i++)
  {
```

Program  
It



# Possible Solutions: top-down flow





# Interactive Tools

## Communication/Computation Profiling and Optimization

Benchmark - interesting.c - Eclipse SDK

File Edit Refactor Navigate Search Project Run Window Help

S: single\_system\_ P: interesting C: le\_32 T: Debug Build Active Run Profile Debug

Profile Results  
Auto-interesting-le\_32-Debug

```
787 table[i] += 1;  
194 if ((num_queues > 0) && (num_entries > 0)) {  
    for (k = 0; k < num_entries; k++) {  
        if (is_producer) WRITE_Q1(table[i]);  
        else read_dummy = READ_Q1();  
        if (num_queues > 1) {  
            if (is_producer) WRITE_Q2(table[i]);  
            else read_dummy = READ_Q2();  
        }  
    }  
}
```

Count	Address	Instruction
0	60000f40	addx4 a15, a15, a2
0	60000f43	l32.n a15, a15, 0
0	60000f45	write_q1 a15
0	60000f48	j 60000f51 <main+0x1d1>
		main+0x1cb
		main+0x1cc
		else read_dummy = REA...
0	60000f4c	read_q1 a3
0	60000f4f	s32.n a3, a1, 28
0	60000f51	if (num_queues > 1) {
		l32.n a5, a1, 4

Console Profile Call-Graph Comparison Saved Output Pipeline Event Trace

Test Graph

162.90 %



# System-Level Design Methodology

System Specification

Abstract task model

Map tasks to processors

Select communications mode

**Compute**

**Communication**

Automatically tune processor

Automatically optimize interconnect

Automatically generate accurate system model.

Detailed application development

VLSI implementation

**Software**

**Hardware**

Integrate software

Fabricate prototype

Bring up full system on final chip

Common SW tools  
across all stages

```
Dot.C | DotProduct.be | Dot.C | npsys.nmap | XTMPMainTemplate.c | C:\OutDir
Xtensa Explorer GENERATED MAIN!
This XTMP_main cannot be compiled in Xtensa Explorer. You must
it into the appropriate environment for host compilation.

Further, you should scan the file for two things. First, you
sanity check to make sure that your system looks right. Second,
will in some cases not be able to generate a complete XTMP_
such a case occurs you will see a comment noting that in the
below.

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "tes/sp.h"

static void loadPrograms( XTMP_core *cores, int numProc );
static int initCoresFromFile( FILE *fp, XTMP_core *cores, XTMP_
// number of processors
#define NUM_PROCESSORS 2
int XTMP_main(int argc, char **argv)
{
    XTMP_core cores[NUM_PROCESSORS];
    XTMP_params params[NUM_PROCESSORS];
    XTMP_multiAddressMapConnector router;
    XTMP_memory *memories;
    unsigned int dontcare = 0x0; /* set addresses with mapEnt
int i = 0;
while( i < argc )
```

## Impact of unified MP architecture

1. Essential base code compatibility across all processors
  2. Common tools, models across all processors
  3. Seamless mix-n-match of features and performance optimization
- *Reduced design effort*
  - *Reduced design risk*
  - *Increase silicon and platform reuse*

