Luthier: Bridging Auto-Tuning and Vendor Libraries for Efficient Deep Learning Inference

Yongin Kwon[1,2], JooHyoung Cha[2], Sehyeon Oh[2], Misun Yu[1], Jeman Park[1], Jemin Lee[1,2*]

[1] *Electronics and Telecommunications Research Institute (ETRI)*
[2] *University of Science and Technology (UST)*

International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES) 2025

September 29, 2025

Importance of Deep Learning Inference Optimization

- **Widespread Applications of Deep Learning**
  - Image analysis, speech recognition, NLP
  - Increasingly integrated into daily life
- **Hardware Constraint Scenarios**
  - Limited by cost, energy, and physical space
  - Need to leverage conventional CPUs or new hardware
- **Determinants of Inference Efficiency**
  - Deep Learning Compilers (AutoTVM, Ansor)
  - Inference Libraries (ArmNN, XNNPack, ONNXRuntime)

Problems with Existing Approaches

Auto-tuning Compilers

- ✓ High flexibility
- ✓ Amenable to automation
- × Very long tuning time (tens of hours)
- × **Still lack support for asymmetric multicores**

Vendor Libraries

- ✓ Immediate execution
- ✓ Optimized kernels
- × Lack of flexibility
- × Difficult to adapt to emerging models

Core Problem

No research considers both optimal kernel selection and workload distribution on asymmetric multicore processors

Luthier: Our Solution

- **Goal**: Combine advantages of auto-tuning and vendor libraries
- **Key Ideas**:
  - Leverage vendor-optimized kernels
  - Fast optimization with ML-based cost model
  - Workload distribution for asymmetric multicores
- **Main Achievements**:
  - Execution speed: Up to **2.0x** improvement
  - Tuning time: **95%** reduction
  - Support for diverse platforms (CPU, GPU)

Optimization Space Complexity

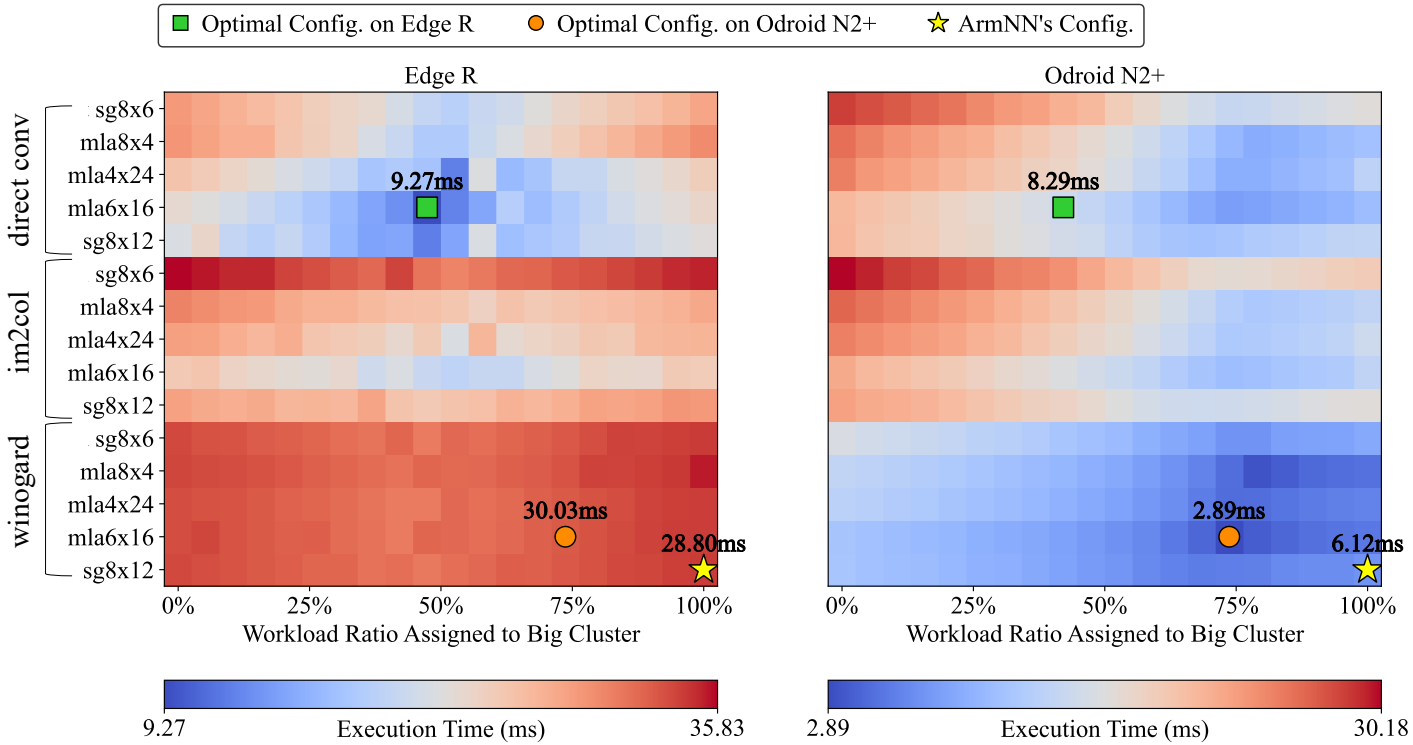

Figure: Heatmap of execution time for ResNet18's 2nd conv layer

- Different kernels and workload distributions show varied performance
- Optimal configuration differs between Edge R and Odroid N2+
- ArmNN's default setting often suboptimal

Performance Differences in Asymmetric Multicores

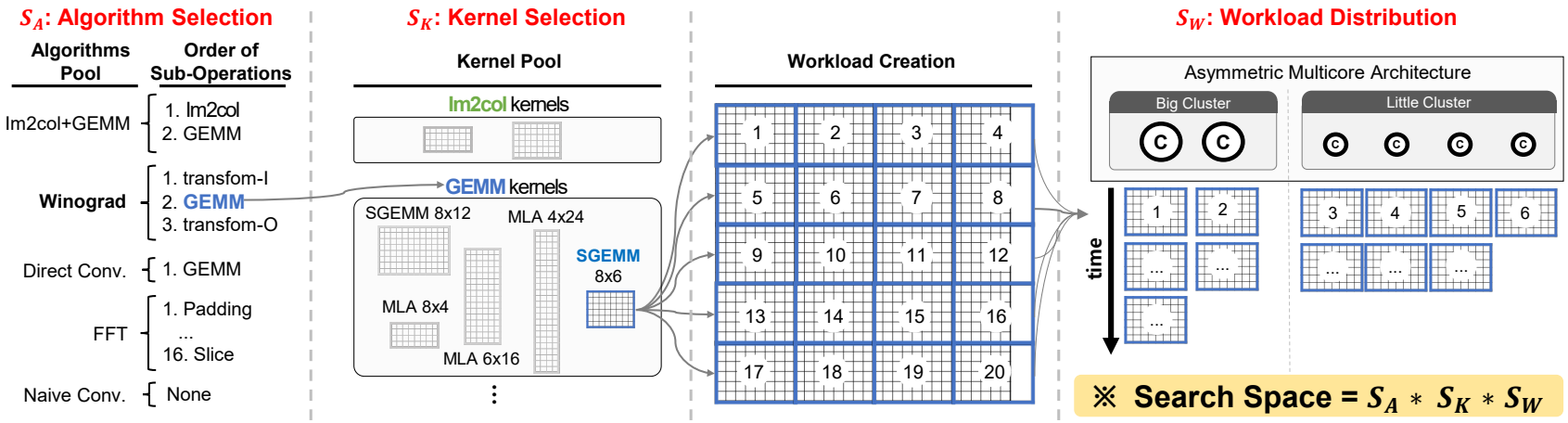

Figure: Workflow of convolution operations on asymmetric multicore architecture

- Same ISA but different optimal execution per hardware implementation
- Performance characteristics vary with core cluster combinations
- Algorithm selection (S_A), kernel selection (S_K), workload distribution (S_W)
- GEMM kernels specialized for A53/A55, hand-optimized in assembly
- Workload distribution is the most critical tuning knob

Importance of Workload Distribution

Why existing approaches fail to optimize workload distribution

AutoTVM & Ansor

- × Focus only on kernel code tuning
- × Runtime creates fixed threads (= big cores)
- × Workload by sub-task index modulation
- × **Cannot dynamically adjust distribution**

Inference Libraries

- × Support wide range of ops/hardware
- × Need fast runtime decisions
- × Static rules (threads = big cores)
- × **Predetermined rules → suboptimal**

Luthier's Solution

ML-based dynamic optimization for both optimal kernel selection and workload distribution across asymmetric multicores

Luthier System Components

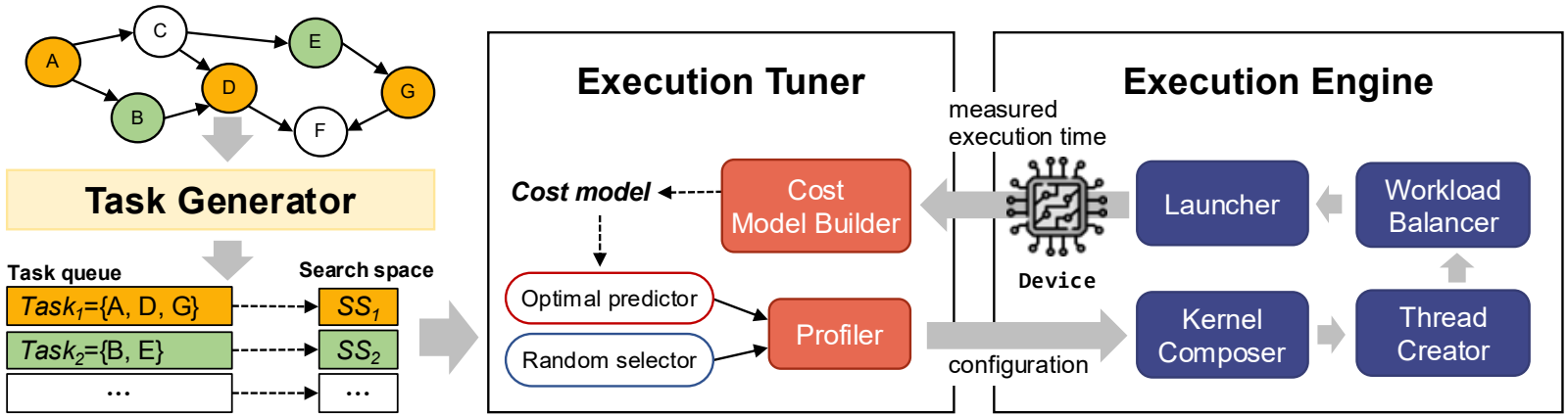

Figure: Overall tuning process of Luthier

Three Main Components:

- **Task Generator**: Generates task queue, groups operations (ResNet101: 104→23 tasks)
- **Execution Tuner**: Determines optimal configuration using XGBoost cost model
- **Execution Engine**: Runs optimized configuration with workload distribution

Tuning Knobs

Knob	Description	Values
Algorithm	Algorithm to form execution kernels	Winograd, Direct, ...
Kernel	Type of GEMM kernel and shape	mla8x4, dot16x8, ...
Workload Distribution	Ratio of workloads for each cluster (Big, Middle, Little)	Big: 0-100% Middle: 0-100% Little: 0-100%

- **Algorithm and Kernel**: Straightforward selection
- **Workload Distribution**: High complexity, expands search space
- **Search Space**: Reduced through task grouping

Experimental Environment

Test Platforms:

- Edge R (Cortex-A72 + Cortex-A53, Mali-T860MP4 GPU)
- Odroid N2+ (Cortex-A73 + Cortex-A53, Mali-G52 GPU)
- Snapdragon 865 (3-cluster Kryo CPU, Adreno 650 GPU)

Baselines:

- Libraries: ArmNN, XNNPack, ONNXRuntime, TFLite
- Auto-tuners: AutoTVM, Ansor

Test Models:

- Vision: ResNet, MobileNet, VGG, etc.
- Language: BERT (encoder), GPT (decoder)

Hardware Specifications

Cluster	Edge R	Odroid N2+	SD865
		Big Cluster	
μArch	A72	A73	Kryo 585 Gold
# Cores	2	4	1
Frequency	1.8 GHz	2.4 GHz	2.84 GHz
		Middle Cluster	
μArch	-	-	Kryo 585 Gold
# Cores	-	-	3
Frequency	-	-	2.42 GHz
		Little Cluster	
μArch	A53	A53	Kryo 585 Silver
# Cores	4	2	4
Frequency	1.4 GHz	2.0 GHz	1.8 GHz
		Mobile GPU	
μArch	Midgard	Bifrost	Adreno 600 Series
# Cores	4	6	2
Frequency	800MHz	800 MHz	587MHz

Performance on Deep Learning Operations

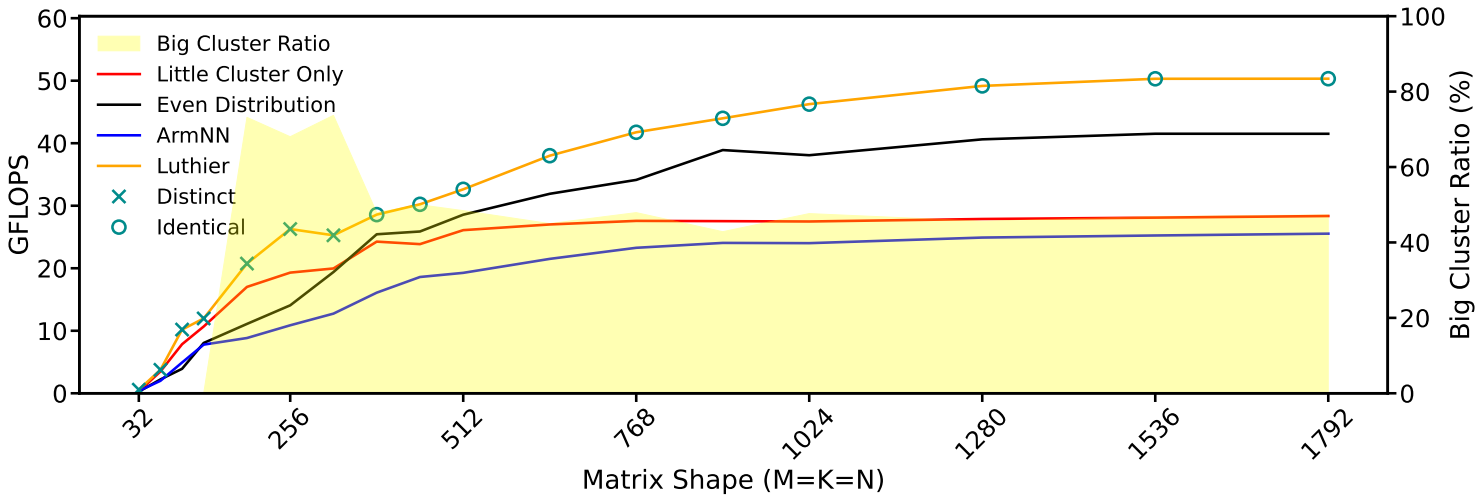

Figure: GEMM performance on Odroid N2+ with different matrix sizes

- Luthier achieves **50 GFLOPS** vs ArmNN's 23 GFLOPS
- Optimal workload distribution varies with matrix size

Convolution Performance Comparison

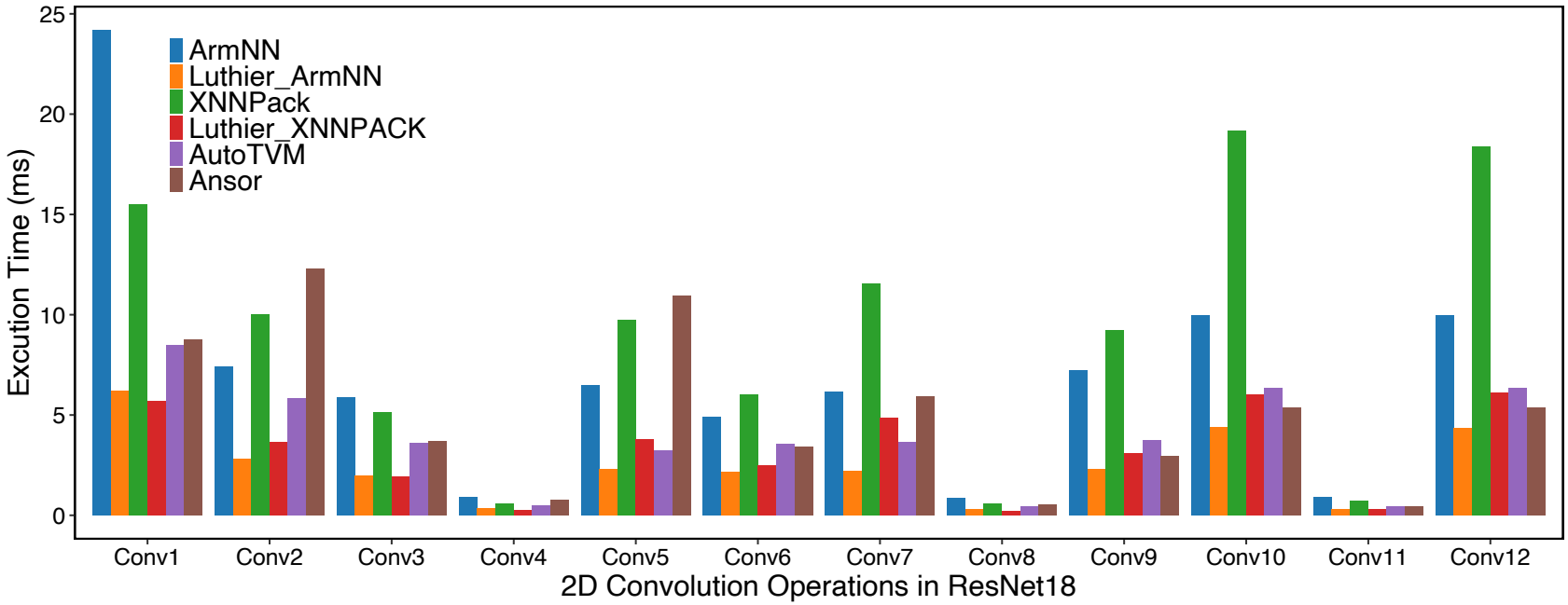

Figure: Performance comparison on convolution operations

- Luthier shows **2.8x speedup** over ArmNN (geometric mean)
- **2.6x speedup** over XNNPack
- In most cases, surpasses AutoTVM and Ansor

Performance Comparison Summary

	Edge R			Odroid N2+		
Model	**Luthier**	AutoTVM	Ansor	**Luthier**	AutoTVM	Ansor
ResNet18	**1.4x(1.6h)**	1.3x(27.1h)	0.8x(29.2h)	**1.7x(1.6h)**	1.9x(10.1h)	1.4x(26.0h)
ResNet50	**1.6x(2.0h)**	1.2x(19.5h)	1.1x(27.1h)	**2.0x(1.9h)**	1.8x(6.2h)	1.5x(27.1h)
ResNet101	**1.6x(2.7h)**	1.1x(51.5h)	1.0x(46.2h)	**2.0x(2.6h)**	1.7x(23.2h)	1.4x(34.7h)
AlexNet	**1.6x(0.4h)**	0.8x(33.4h)	0.6x(32.7h)	**1.3x(0.4h)**	0.6x(53.7h)	1.4x(25.8h)
VGG16	**1.5x(0.9h)**	1.0x(33.3h)	0.5x(30.4h)	**1.6x(0.8h)**	1.2x(31.0h)	0.8x(25.8h)
GoogLeNet	**1.4x(2.4h)**	1.3x(49.8h)	1.1x(28.5h)	**1.6x(2.3h)**	1.6x(22.6h)	1.5x(23.2h)
MobileNetV2	**1.1x(1.6h)**	1.7x(43.2h)	1.8x(62.7h)	**1.5x(1.5h)**	1.8x(49.8h)	2.8x(23.8h)
Average	**1.5x(1.7h)**	1.1x(39.2h)	0.9x(36.5h)	**1.7x(1.7h)**	1.4x(28.5h)	1.5x(27.0h)

- Luthier consistently achieves best speedup with minimal tuning time
- Speedup values shown relative to ArmNN baseline (in parentheses: tuning time)

Transformer Models Performance

Language Model Results:

- BERT (encoder-only): Up to **1.8x** speedup
- GPT (decoder-only): Up to **1.8x** speedup

Key Achievement

Superior performance compared to AutoTVM and Ansor while reducing tuning time by 95%

Mobile GPU (Mali-G52) Results:

- Performance improvement even in symmetric core environments
- Meaningful optimization achieved through kernel selection alone

Impact of Tuning Knobs

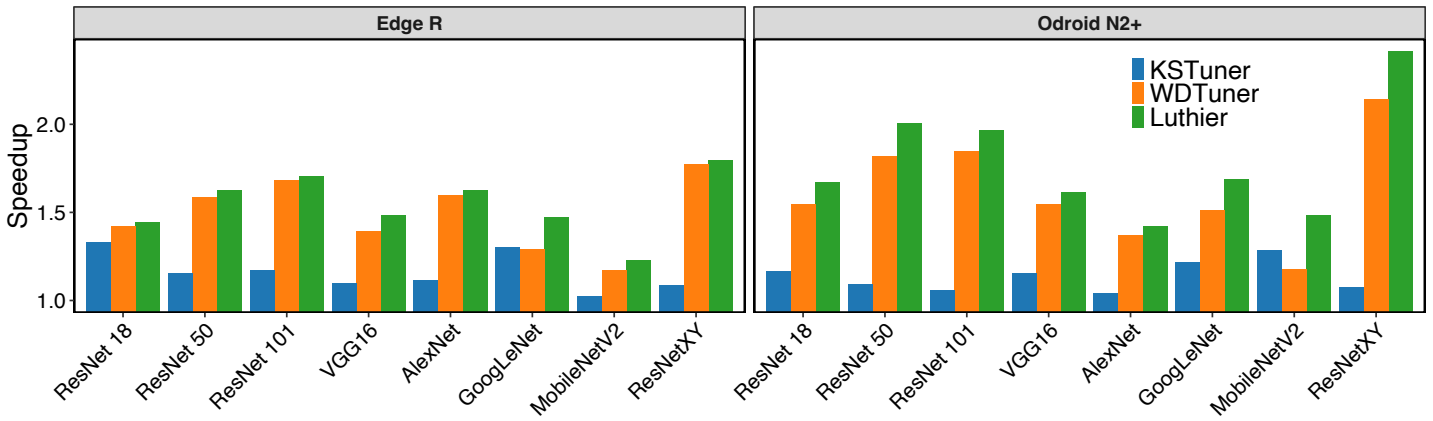

Figure: Speedup by Kernel Selection (KS) and Workload Distribution (WD)

- Both kernel selection and workload distribution crucial
- Combined tuning yields best performance

Scalability and Compatibility

Support for Various Libraries:

- Successfully applied to ArmNN and XNNPack
- Integration possible with minimal code modifications

Hardware Platform Scalability:

- Supports from big.LITTLE to 3-cluster configurations
- Optimization for both CPU and GPU

Future Research Directions:

- Integration of additional libraries (e.g., ONNXRuntime)
- Support for deep learning accelerators
- Optimization for Large Language Models (LLMs)

Conclusion

Main Contributions of Luthier:

1. Combines vendor library optimized kernels with auto-tuning
2. Optimizes workload distribution for asymmetric multicores
3. Fast tuning based on machine learning

Achievements:

- Execution speed: Up to **2.0x** improvement
- Tuning time: **95%** reduction
- Support for diverse hardware and models

Conclusion

Luthier provides a practical and efficient deep learning inference optimization solution, demonstrating excellent performance particularly in asymmetric multicore environments