Performance Enhancement under Power Constraints using Heterogeneous CMOS-TFET Multicores

Emre Kultursay, Karthik Swaminathan, Vinay Saripalli
Vijaykrishnan Narayanan, Mahmut T. Kandemir, Suman Datta
The Pennsylvania State University
{euk139,kvs120,vijay,kandemir}@cse.psu.edu,vxs924@psu.edu,sdatta@engr.psu.edu

ABSTRACT

Device level heterogeneity promises high energy efficiency over a larger range of voltages than a single device technology alone can provide. In this paper, starting from device models, we first present ground-up modeling of CMOS and TFET cores, and verify this model against existing processors. Using our core models, we construct a 32-core TFET-CMOS heterogeneous multicore. We then show that it is a very challenging task to identify the ideal runtime configuration to use in such a heterogeneous multicore, which includes finding the best number/type of cores to activate and the corresponding voltages/frequencies to select for these cores. In order to effectively utilize this heterogeneous processor, we propose a novel automated runtime scheme. Our scheme is designed to automatically improve the performance of applications running on heterogeneous CMOS-TFET multicores operating under a fixed power budget, without requiring any effort from the application programmer or the user. Our scheme combines heterogeneous thread-to-core mapping, dynamic work partitioning, and dynamic power partitioning to identify energy efficient operating points. With simulations we show that our runtime scheme can enable a CMOS-TFET multicore to serve a diversity of workloads with high energy efficiency and achieve 21% average speedup over the best performing equivalent homogeneous multicore.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: General—*Hardware/software interfaces*; C.1.3 [**Processor Architectures**]: Other architecture Styles—*Heterogeneous (hybrid) systems*

Keywords

Heterogeneous multicores, TFETs, power aware systems, power partitioning

1. INTRODUCTION

Industrial scaling trends [1] show that each step in process technology comes with a larger number of cores that can be fit into a single chip, and yet, the power deliverable to the chip stays relatively constant. The lack of enough power to turn on all the cores simultaneously at their peak frequency manifests itself either as *dark silicon* where only a small number of cores can be turned on, or as *dim silicon* where all the cores are turned on but execute at very low voltages/frequencies [2].

CODES+ISSS'12, October 7-12, 2012, Tampere, Finland.

Depending on the characteristics of the application to be executed on a multicore, the answer to which of the two possible configurations, i.e., dark silicon or dim silicon, performs better, can vary. Scalable applications may prefer using a large number of cores at a low frequency to achieve a higher energy efficiency. Other applications that do not scale beyond some degree of parallelism would rather be restricted to run on a smaller number of cores (sequential applications being at one extreme of this case, running on only one core). In the latter case, the power saved by turning the remaining cores off can be used to increase the clock frequency of the active cores. Clearly, a general purpose processor with a goal to perform well on a diversity of workloads must be optimized for both scenarios.

Heterogeneous multicores employ separate cores for high performance and low power operation modes. As a result, a heterogeneous multicore has the potential to achieve better power and performance characteristics than an equivalent homogeneous multicore over a variety of workloads. In this work, we first pose the question: *How to design an energy efficient heterogeneous multicore?* We first identify that as the number of cores increase, the dim silicon approach results in very low per-core power budgets. When cores are implemented using the CMOS technology, such a low per-core power budget forces the transistors to operate at near or sub-threshold voltages at which CMOS devices show very poor energy efficiency. Hence, CMOS technology is not very suitable for a dim silicon approach.

TFET technology [3][4] is a very promising alternative to sub-threshold CMOS, offering better energy-delay performance in the low voltage regime. Building TFET cores and employing them in a multicore can significantly improve performance when operating at low per-core power budgets. Yet, TFETs are not energy efficient at higher voltages and perform poorly in a dark silicon setting. At such high voltages, CMOS technology is still the most energy efficient choice. Motivated by these observations, we identify a heterogeneous CMOS-TFET multicore as the best solution to serve a diversity of workloads. However, porting applications to this heterogeneous multicore is not a trivial task. Specifically, it involves finding the best performing thread-to-core mapping, work-to-thread distribution, and core power distribution.

In this work, we propose an *automatic runtime scheme* to extract high performance from heterogeneous CMOS-TFET multicores that operate under fixed power budgets. Our scheme can be implemented as a part of the operating system (OS) and uses (i) heterogeneous thread-to-core mapping (i.e., it can utilize both types of cores simultaneously), (ii) dynamic work partitioning (i.e., it distributes work unequally across threads), and (iii) dynamic power partitioning (i.e., it distributes power unequally across cores). This scheme analyzes the efficiency of the application threads running on the target heterogeneous multicore and redistributes

the available chip power across cores to improve overall performance. Our specific contributions in this work are:

• We identify device-level heterogeneous CMOS-TFET multicores as a promising approach to effectively serve a diversity of workloads.

• We build a core power model starting from device models. We validate this model by comparing its results with data reported for existing processors. We use this model with the emerging FinFET and TFET technologies and construct a CMOS-TFET heterogeneous multicore power model.

• We evaluate a CMOS-TFET multicore under a fixed power budget and compare it against homogeneous CMOS and homogeneous TFET multicores. We show that with simple, homogeneous (i.e., single core type) thread-to-core mapping and static work partitioning, a heterogeneous multicore can achieve performance comparable to the best performing homogeneous multicore.

• We propose an automatic runtime scheme that performs dynamic power partitioning across different types of cores in the CMOS-TFET heterogeneous processor. This scheme considers the power/performance tradeoffs and quantities of different types of cores, and can improve the performance of a variety of applications running on heterogeneous processors with different configurations. We show that, when used with heterogeneous thread-to-core mapping and dynamic work partitioning, this runtime scheme enables our heterogeneous CMOS-TFET multicore to achieve an average performance improvement of 21% over the best performing homogeneous multicore, without any effort from the programmer or the user.

2. THE CASE FOR A HETEROGENEOUS CMOS-TFET MULTICORE

In this section, we demonstrate the need for heterogeneous architectures to cater to a diversity of applications. We focus our attention on device-level heterogeneity in multicore processors and examine the viability of using Tunnel-FET based cores in processor design.

2.1 Power Constrained Multicores

The ITRS roadmap [5] indicates that the maximum allowable chip power will remain constant over the next several generations even amidst technology scaling. This makes the chip power the most important parameter that constrains the maximum performance that can be extracted from a processor. Improving energy efficiency has become a major goal, which has lead to servers-on-chip with an entire power budget of a few Watts have been built [6] for web applications as well as 'big data' applications like MapReduce and Hadoop. The corresponding per-core power comes out to be less than 1W. power budgets corresponding to less than 1 Watts per-core. Under such restricted power budgets, the number of active cores and their frequencies must be carefully determined to preserve efficiency. Figure 1 shows the power consumption of *swim* from the SPEC-OMP benchmark suite [7] using an 8-core and a 32-core multicore operating over a range of frequencies. It is observed that the power demand for 32 cores to run at 2 GHz, which is around 120W, far exceeds a power budget of 32W (1W per core), which is barely sufficient to turn on 8 cores running at 2 GHz. Thus the only way of reaching 2 GHz frequency with the 32 core processor is by completely shutting down at least 24 cores, i.e, 24 cores must remain *dark*. Conversely, we see that in order to be able to use all the 32 cores, the clock frequency must be limited to less than 1 GHz, i.e, the cores are run in a *dim* manner. This dark/dim behavior is common to many other applications as well (e.g., *apsi* and *wupwise*). These scenarios represent the two extreme operating points, and which one of the two achieves higher performance depends on the properties of the target application.

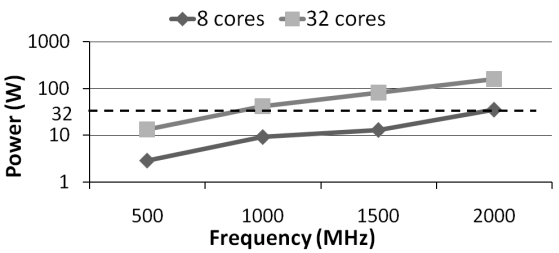

Figure 1: Power consumption of *swim* using 8 and 32 cores at a frequency range of 500 MHz to 2 GHz. The dashed line represents a power budget of 32 W.

2.2 Serving a Diversity of Applications

In order to efficiently map applications onto multicore processors under power constraints, two properties of the target applications must be studied: (1) scalability of the application with number of cores, and (2) scalability of the application with frequency.

Figure 2 compares the scalability of two applications, *swim* and *equake*, with the number of cores. It can be observed that while *swim* scales well even when run on 32 cores, the speedup of *equake* is limited when executed on more than 8 cores, in fact degrading when the number of cores is increased from 16 to 32. Hence, *equake* is not able to utilize the cores allocated to it as effectively as *swim*. Analyzing the scalability of two applications with frequency, Figure 3 shows the speedup of two applications, *swim* and *gafort*, when they are executed on a 32-core processor at a range of frequencies. As frequency is increased, the speedup achieved by *gafort* is higher than *swim*. In general, applications with a higher resource utilization in the cores see a higher benefit from increased core clock frequency.

These application characteristics can be used to find better operating points (number of cores, frequencies) that *improve performance under a given power budget*. For instance, given a power budget of 1W/core (32W overall) and considering a baseline configuration with a single CMOS core running at 500 MHz, a well scaling application like *swim* achieves its best performance when running on 32 cores at 1 GHz to give a 52X speedup. On the other hand, *equake*, which scales poorly, achieves its optimal speedup of 15X at 2 GHZ with 8 cores. This indicates that different multithreaded applications can prefer different operating points to maximize their performance under a power budget.

2.3 Exploiting Device-Level Heterogeneity

The best configuration for running a particular application can be significantly different depending on the properties of the target application. In the dark silicon case, cores must be optimized to be efficient at high frequencies, and in the dim silicon case, they must be optimized for low frequency operation. As these two requirements are at the two ends of the power-performance spectrum, they *cannot* be satisfied by a single device technology simultaneously. On the other hand, it is possible to combine two technologies in implementing a multicore, where one technology is used to implement low-frequency/low-power cores and another technology is used to implement high-frequency/high-performance cores. This heterogeneous processor can be a common platform to serve a diversity of applications with

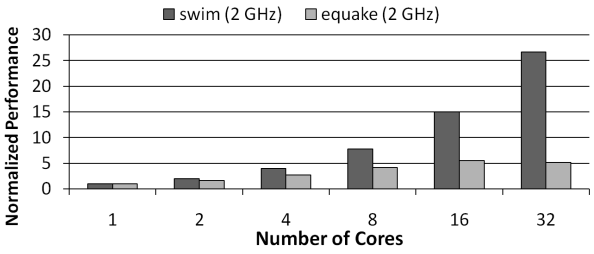

Figure 2: Scaling of the parallel regions of *swim* and *equake* under various core counts for a system running at 2 GHz. Performance values are normalized with respect to the 1 core case.

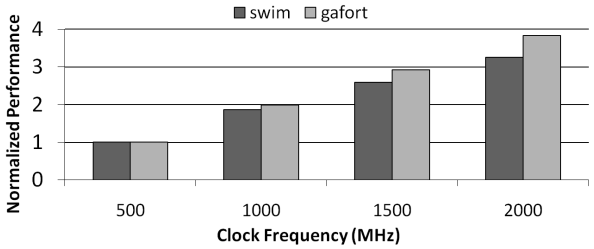

Figure 3: Scaling of applications *swim* and *gafort* with clock frequency for a 32 core system. Performance values are normalized with respect to the 500 MHz case.

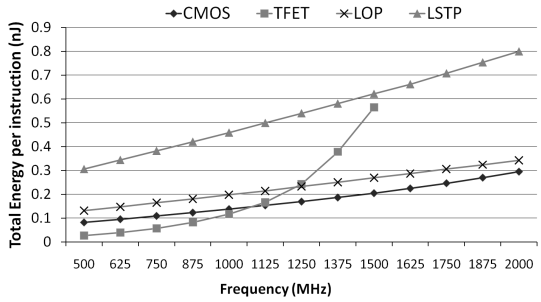

Figure 4: Variation in dynamic energy with frequency for different transistor technologies.

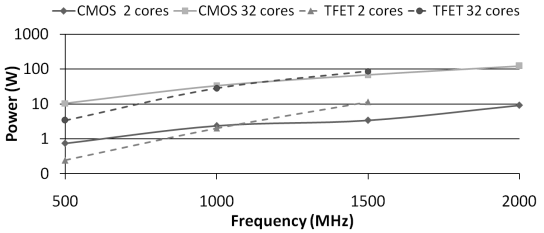

Figure 5: Variation in CMOS and TFET processor power at different frequencies for different number of cores.

high performance. While the standard CMOS technology that can reach high frequencies is a good candidate to be used for implementing high-performance cores, its low supply voltage ($V_{cc} < 300mV$) operation is extremely inefficient because of its limited overdrive voltage. Under a small overdrive voltage, CMOS devices are limited by a 60 mV/decade minimum sub-threshold slope, which provides a very low drive current. Stronger drive currents at low supply voltage can be achieved by reducing the CMOS threshold voltage (V_t). However, reduced threshold voltage results in an unacceptable increase in off-state leakage current. Therefore, the sub-threshold slope of CMOS technology results in poor I_{on}/I_{off} ratio at low voltages, leading to an energy-inefficient low frequency operation.

Variants of standard high-performance (HP) CMOS devices are low standby power (LSTP) and low-power (LOP) devices. Figure 4 shows the variation in core dynamic energy with core frequency for these CMOS-based transistor technologies. LSTP transistors have a higher oxide thickness than HP CMOS transistors which leads to lower I_{on} as well as I_{off} . While this decreases the leakage power, a higher V_{cc} is needed for an LSTP transistor to operate at the same clock frequency as an HP transistor, which in turn results in a higher dynamic energy consumption. Similarly, using LOP transistors which operate at a lower V_{cc} but have a higher threshold voltage also cannot reach better energy efficiencies at any frequency. As a result, using LSTP or LOP transistors is not an effective solution for the design of dynamic energy dominated cores.

Any solution to implement energy-efficient cores must clearly have a better sub-threshold behavior than CMOS technology. Interband Tunnel Field Effect Transistor (TFET) technology is a very promising alternative to sub-threshold CMOS, showing a characteristic sub-60 mV/decade sub-threshold slope. This steep slope brings a good I_{on}/I_{off} ratio at low voltages, which translates into high drive current and low off-state leakage current. Thus, TFET-based cores are clearly more energy efficient than any of the other technologies at low frequencies, as shown in Figure 4.

Figure 5 shows the average power consumption of the *swim* application when running on both CMOS and TFET-based homogeneous multicores with 8 and 32 cores. At the lowest frequency of 500 MHz, 32 TFET cores consume $3X$ less power than 32 CMOS cores. When working under a fixed power budget, this can translate into more number of cores being turned on. Consequently, the massive parallel computational capacity brought in by 32 TFET cores can easily outperform the CMOS configuration when running a scalable application such as *swim*. At low voltages, it is also possible to operate the TFET at higher clock frequencies than CMOS. This enables the TFET cores to achieve superior performances than CMOS cores under power constraints for highly scalable applications.

When executing poorly scaling applications or sequential parts of parallel applications, it is better to restrict the entire power budget to a small number of cores running at a high operating frequency. The high-frequency optimized CMOS cores are very suitable for this operating point. From our profiling simulations we observed that, given a power budget of 32W, we can typically run only 8 CMOS cores at a peak frequency of 2 GHz. As a result, we designed a 32 core heterogeneous CMOS-TFET architecture with 8 CMOS cores and 24 TFET cores. This ensures that, when needed, 8 CMOS cores can be executed at the maximum frequency.

3. MODELING CMOS AND TFET CORES

In this section, we first present our FinFET and TFET device models, and then, describe the method we use to construct an entire processor model from these devices.

3.1 Device Modeling

There have been numerous experimental demonstrations of novel tunneling (TFET) devices. The first such demon-

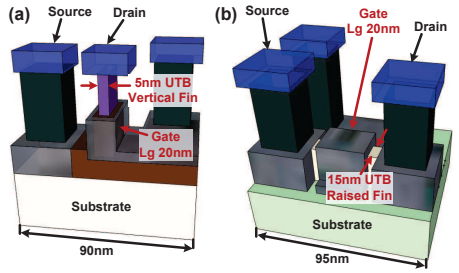

Figure 6: (a) Structure of an UTB 3D TFET, (b) structure of a 3D Si FinFET.

stration was an $In_{0.53}Ga_{0.47}As$ homojunction TFET [8] which illustrated the concept of a vertical interband tunneling transistor. The problem of low I_{on} in homojunction TFETs was overcome by using a $GaAs_{0.1}Sb_{0.9}/InAs$ heterojunction TFET. On account of the P-N heterojunction being staggered, and InAs having a lower bandgap (E_G), this device was observed to have a higher I_{on} than a homojunction TFET. Recently, a vertically-oriented, gate-all-around silicon nanowire was demonstrated, showing 50 mV/decade sub-threshold slope over 3 decades of drain current [9], thus experimentally illustrating a sub-60 mV/decade slope. Further, a process flow for the creation of a side-gated vertical-mesa TFET which can be scaled down to achieve an Ultra-Thin-Body (UTB) double-gated structure has also been demonstrated [3, 10]. Continuous improvements in vertical 3D UTB heterojunction tunneling structures are bringing us closer to the promise of sub-60 mV/decade sub-threshold slope operation.

Figure 6(a) shows the 3D schematic of a vertically-oriented UTB TFET, the fabrication of which has been demonstrated in [10]. Modeling of such a transistor has been performed for different materials using advanced atomistic simulations [11, 12]. This TFET device is compared against bulk-CMOS devices in 20nm CMOS technology in [4]. However, based on recent trends [13, 14], it has become clear that beyond 20nm technology node, instead of bulk-CMOS, CMOS transistors will be implemented using the tri-gate transistor technology (i.e., FinFETs). The 3D schematic of a silicon FinFET is shown in Figure 6(b). The experimental demonstration of such a transistor for the 20nm technology node has been shown in [15]. In this work, we work with these emerging heterogeneous TFET and silicon FinFET transistors.

We performed device level modeling of TFET devices using TCAD Sentaurus [16]. TCAD simulations for TFETs are calibrated with the atomistic simulations [11, 12], the results of which are shown in Figure 7(a) together with the device parameters used. The steep slope in this figure shows that radically different device physics of TFETs allows departure from 60 mV/decade limit imposed by nature on conventional devices such as BJTs and MOSFETs. We verified that our TCAD simulation results agree with the experimental TFET data reported in [3, 10]. Similarly, we performed TCAD simulations for the 20nm Si FinFET. Figure 7(b) shows that our TCAD simulation of this device is in close agreement with the experimental data shown in [15].

Using our device models, we obtained the performance data for a 20nm high performance (low V_t) Si FinFET NMOS and a 20nm heterojunction n-channel TFET, which are shown in Figure 8. This figure shows the I_{on} versus I_{on}/I_{off} for the two types of transistors for different operating points along the I_d-V_g curve, for a given V_{cc} window, and at two supply voltages, namely, $1V$ (high voltage) and $0.3V$ (low voltage). From this figure, we observe that at $1V$, the Si FinFET outperforms the heterojunction TFET both in I_{on} as well as I_{on}/I_{off} ratio. However, when the supply voltage is reduced to $0.3V$, the TFET becomes better in I_{on} and the I_{on}/I_{off} ratio, compared to the high performance Si FinFET at $0.3V$. Even using a low-power FinFET with high V_t exhibits a similar behavior, because TFETs are much more energy efficient at low voltages on account of their sub-60mV sub-threshold slope. While TFET is preferred at low V_{cc} values less than $0.3V$, CMOS is preferred at high V_{cc} values of around 1V. Table 1 summarizes the performance of the two types of transistors at $1V$ and $0.3V$.

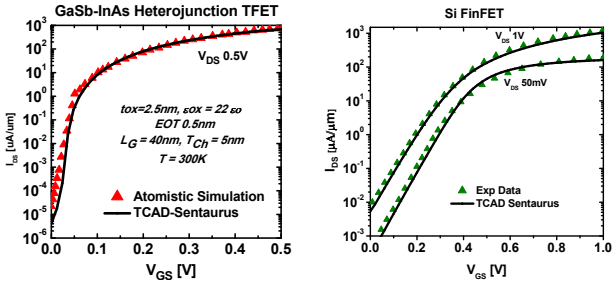

Figure 7: (a) Comparison of TCAD simulation with atomistic simulation for heterojunction TFET [11]. (b) Comparison of TCAD simulation with experimental data for Si FinFET [15].

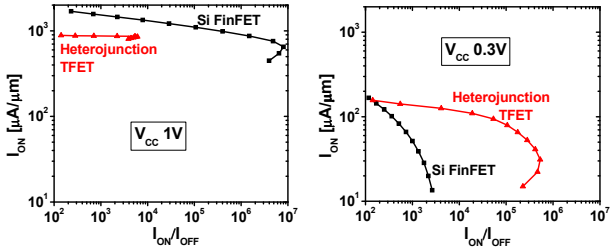

Figure 8: I_{on} vs I_{on}/I_{off} ratio comparison for Si FinFET and HTFET at V_{cc} of 1V and 0.3V.

Table 1: Summary of transistor performance.

Tech	V_{cc}[V]	I_{on}[uA/um]	I_{off}[nA/um]	I_{on}/I_{off}
FinFET	1	1200	5	2×10^5
TFET	1	850	150	6×10^3
FinFET	0.3	5.5	1.25	4.5×10^3
TFET	0.3	115	6	2×10^5

3.2 Transistor-to-System Abstraction

Processor-level power-performance comparison for CMOS and TFET-based processors, including simulations of memory and datapath elements, was done using a device-to-system architecture abstraction in [4]. We use a similar abstraction with two key enhancements in modeling: (1) our TFET models are calibrated with atomistic simulations and experimental data, and (2) our CMOS models use an advanced tri-gate (FinFET) technology rather than an older bulk-CMOS technology.

The maximum operating frequency of a processor is set by its critical path, which we model using a ring-oscillator chain of NAND gates each driving five NAND gates, as shown in Figure 9(a), resulting in a 45 FO4 delay for the ring-oscillator. We observe that, using a 45nm bulk-CMOS model [17], this logic chain is able to accurately model the

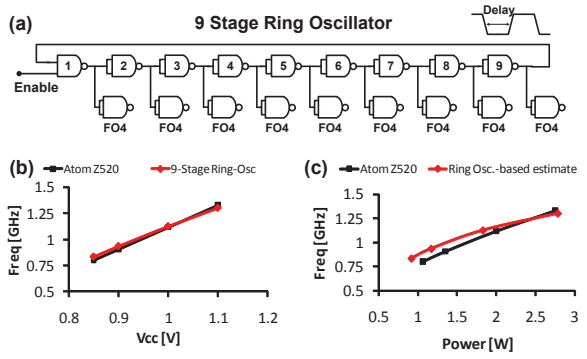

Figure 9: (a) 9-stage ring-oscillator used to model the critical path in the processor, (b) frequency vs V_{cc} behavior for the Atom processor, and (c) frequency vs power behavior for the Atom processor.

V_{cc}-frequency behavior of the Intel Atom processor [18], as shown in Figure 9(b). The switching energy per transistor is estimated from the ring oscillator. Typical switching factors obtained during logic benchmarking range from 12.5% (in logic intensive designs) to 50% (in arithmetic intensive designs), making an average activity factor of 30% a reasonable assumption [19]. Using the Atom processor's transistor count of 47 million and this 30% transistor switching activity factor, we were able to reproduce the power-frequency characteristics of the Atom processor.

In order to model a forward-looking processor, we moved from the 45nm technology node to 20nm technology node in our model. Assuming an average IPC of 1, which can be representative of the 2-issue Intel Atom processor architecture, we obtained the energy per instruction of our target processor using 20nm Si FinFET and 20nm heterojunction TFET technology, considering the energy per instruction to be largely independent of the specific instruction being executed [20]. In our evaluation, we considered both the dynamic and leakage energy of the processor.

As observed in Figure 4, on account of the steep subthreshold slope, TFETs deliver better performance compared to Si FinFETs in low-V_{cc} operation ($\leq 0.5V$). Since the tunneling current saturates for higher Vcc ($> 0.5V$) values, TFET processors are less efficient than FinFET based processors at frequencies in excess of around 1.1 GHz. Due to this behavior, there is an energy-delay crossover point, below which the TFET processor is more energy efficient and above which the Si FinFET processor becomes the energy efficient choice.

3.3 The Manufacturing Barrier: Integrating TFET and FinFET Technologies

TFET technology is at an exploratory stage and it is too premature to analyze it from the yield and cost perspectives. However, from an integration perspective, TFET and FinFET devices can be fabricated on a single chip in a monolithic fashion using a silicon substrate. Such integration has already been demonstrated through experimental fabrication of indium gallium arsenide (InGaAs) based Quantum-Well FETs (QWFETs) on silicon substrate using a metamorphic buffer layer growth scheme [21]. Another important parameter in the adoption of new technologies is their sensitivity to process variations. In [22], it is shown that TFET and FinFET devices are affected similarly from process variations. Their work also presents examples of process variation-aware TFET SRAM cell design.

4. PORTING APPLICATIONS TO HETEROGENEOUS MULTICORES

When executing a multi-threaded application on a CMOS-TFET heterogeneous multicore, there are two important questions that must be answered: (1) how will the threads be mapped to cores, and (2) how will the work be partitioned across threads? The performance of the target application can significantly vary based on these decisions, yet analyzing and evaluating alternative schemes to make the right decision is a time consuming task. In this section, after presenting alternative mapping and work partitioning schemes, we propose an automatic scheme that can, without any programmer effort, achieve high performance, by answering a third question: *How should the available power be partitioned across cores?*

4.1 Thread-to-Core Mapping

We first examine how the threads of a multi-threaded application should be mapped to the cores of a heterogeneous multicore. Considering a CMOS-TFET heterogeneous multicore, two possible methods are homogeneous mapping and heterogeneous mapping. Homogeneous mapping assumes that, while running a parallel region of an application, all threads will use the same type of core. In this case, when a parallel region starts, a binary decision is performed and the parallel region is executed either only on CMOS cores or only TFET on cores. This mapping strategy is used by traditional accelerator-based systems (e.g., a host processor with GPGPUs). However, the decision of what type of core leads to better performance under a fixed power budget must be made manually and is a non-trivial task.

An alternative way of mapping application threads to a heterogeneous processor is to run the application on both types of cores simultaneously. However, cores of different types will be executing at different operating points, which can cause significant performance discrepancy. Therefore, heterogeneous thread-to-core mapping can be expected to perform well only when used together with a heterogeneity-aware work partitioning scheme.

4.2 Work Partitioning

The simplest way of partitioning work across the threads of an application is to distribute the total work equally. This method is typically preferred in homogeneous systems where all application threads will be executed on the same type of core. Running on the same core type results in all threads completing their computations at about the same time, minimizing the time during which cores sit idle at synchronization points. On a loop-level parallelized system, static work partitioning distributes work statically at compile time, by assigning equally sized chunks of loop iterations to all threads.

In a heterogeneous multicore, the performance of threads running on different types of cores can be quite different under a heterogeneous thread-to-core mapping. Under static work partitioning, it can take some threads significantly more time to complete their computations. In this case, threads that complete their work early are forced to wait idle until all the other threads reach the synchronization point, which leads to poor processor utilization, and eventually, poor overall performance. One solution is to use *dynamic work partitioning*, the most widely used form of which is *dynamic loop scheduling* [23]. In this scheme, iterations of a parallelized loop are not distributed equally across threads. Instead, each thread executes a different number of iterations proportional to its dynamic performance. As a result,

```
PartitionPower(interval, pct, CorePower, N_CMOS, N_TFET)  :
IPS_old ← 0
direction ← +1
while ( true )
    sleep(interval)
    Power_CMOS ← Σ_{i∈CMOS} CorePower[i]
    Power_TFET ← 1 − Power_CMOS
    IPS_new ← Σ_{i∈[0,31]} instructions[i] ∗ frequency[i]/cycles[i]
    if IPS_new < IPS_old
        direction ← −1 ∗ direction
    end if
    if direction = +1
        ΔPower ← Power_CMOS ∗ pct/100
        Power_CMOS ← Power_CMOS − ΔPower
        Power_TFET ← Power_TFET + ΔPower
    else
        ΔPower ← Power_TFET ∗ pct/100
        Power_CMOS ← Power_CMOS + ΔPower
        Power_TFET ← Power_TFET − ΔPower
    end if
    for i ∈ [0, (N_CMOS + N_TFET − 1)]
        if type[i] = CMOS
            CorePower[i] ← Power_CMOS/N_CMOS
        else
            CorePower[i] ← Power_TFET/N_TFET
        end if
    end for
end while
```

Figure 10: The pseudocode for our dynamic power partitioning algorithm.

all threads complete their allocated work at about the same time, maximizing the utilization of the parallel hardware.

4.3 Power Partitioning

In addition to distributing the workload across threads in the form of loop iterations, the way we partition the available chip power across cores can also significantly impact the performance of the system. As the total power is limited, it must be partitioned across cores such that the processor executes at an energy-efficient operating point. The optimum power distribution depends on the mapping and work partitioning policies adopted by the application. In case of homogeneous mapping and equal work partitioning, dividing power equally across all cores results in almost identical thread performance. However, for other mapping and work partitioning schemes, equal power partitioning may not be the best approach. Instead, adjusting the power distribution across cores can lead to more efficient operating points for the heterogeneous multicore.

Consider the combination of heterogeneous thread-to-core mapping and dynamic work partitioning. While the heterogeneous mapping removes the burden of manually finding the best type of core, dynamic work partitioning eliminates the barrier-wait problems due to load imbalance. Note that when two threads of an application are executed on different types of cores under an equal per-core power budget, we can easily calculate how efficiently they are utilizing this power. For instance, if all cores are running at frequencies below 1 GHz, then clearly TFET cores are operating more efficiently than CMOS cores. Further, determining which type of core uses the available power more efficiently is application-dependent and can even change within an application based on the current phase of execution. Therefore, a runtime scheme that (1) identifies the energy efficiencies of different types of cores and (2) dynamically *repartitions* available power across cores to allocate more power to the efficient core type can potentially improve the overall performance.

To improve performance of our CMOS-TFET heterogeneous multicore under a fixed power budget, we employ a *dynamic power partitioning algorithm* based on a "perturb-and-observe" method. Our algorithm considers the TFET cores and CMOS cores as two power domains and partitions the total chip power across these two domains. Figure 10 gives the pseudocode for this algorithm. At every epoch, a fraction of the power is taken from one power domain (e.g., from TFET) to the other domain (e.g., to CMOS). At the end of the epoch, we observe whether this action resulted in an improvement in the total performance in terms of *instructions per second* (IPS). If the performance improves, then we continue to transfer power in the same direction; otherwise, we reverse the direction of power transfer. Using such a dynamic scheme also enables us to react to behavioral variations occurring within the application. Our dynamic power partitioning algorithm takes the epoch length, percentage of power to transfer at each epoch, total available chip power, and the number of cores in either type as input parameters, and hence, is *portable* across various heterogeneities and power budgets. By employing power partitioning together with a heterogeneous mapping and dynamic loop scheduling, our goal is to reach higher performance automatically, without putting the burden of detailed analysis and testing on the programmer or the user.

4.4 Overview of Evaluated Schemes

Table 2 shows the schemes tested in our evaluation. Starting from a baseline homogeneous processor that is either all-CMOS or all-TFET with equal work partitioning and equal power partitioning (*CMOS-Base* and *TFET-Base*), the first step we take is to switch to a 8-CMOS, 24-TFET heterogeneous processor. For each multi-threaded application, the programmer makes the binary decision of running parallel regions of the application on 8-CMOS or 24-TFET cores. Based on the ability of the programmer in making correct thread-to-core mapping decisions, the performance obtained varies. The schemes with the best and the worst mappings are referred to as *Hetero-Manual-Best* and *Hetero-Manual-Worst*, respectively. To explore the benefits of using all cores in the heterogeneous processor simultaneously (as opposed to using only one type at a time), we also evaluate the *Hetero-Simple* scheme which performs heterogeneous thread-to-core mapping. In the *Hetero-DynWork scheme*, on the other hand, we modify the application to be heterogeneous multicore-aware by adopting dynamic work partitioning. Our goal with this scheme is to evaluate the benefits of dynamic work partitioning on the heterogeneous system. Finally, we introduce our dynamic power partitioning scheme (*Hetero-Auto*) which also includes heterogeneous mapping and dynamic work partitioning.

5. EXPERIMENTAL SETUP

5.1 Simulation Infrastructure

We use the Simics full system simulator for running our simulations [24]. We assume a 32 core system consisting of 8 CMOS cores and 24 TFET cores as our heterogeneous processor model. The performance of this system is compared to our *baseline* system consisting of 32 cores (CMOS or TFET). The architectural parameters of the simulated systems are shown in Table 3.

5.2 Dynamic Work and Power Partitioning Implementation

In order to eliminate the effects of load imbalance, we use dynamic parallel loop scheduling which performs work partitioning at runtime. Dynamic loop scheduling can be done in two ways. In one method, the workload chunk size allotted to each thread is fixed. However, this may not be

Processor	Mapping	Work Partitioning	Power Partitioning	Code
32 CMOS cores	Homogeneous: CMOS	Equal	Equal	CMOS-Base
32 TFET cores	Homogeneous: TFET	Equal	Equal	TFET-Base
8 CMOS and 24 TFET cores	Homogeneous: CMOS *or* TFET	Equal	Equal	Hetero-Manual-Best
	Homogeneous: CMOS *or* TFET	Equal	Equal	Hetero-Manual-Worst
	Heterogeneous: CMOS *and* TFET	Equal	Equal	Hetero-Simple
	Heterogeneous: CMOS *and* TFET	Dynamic	Equal	Hetero-DynWork
	Heterogeneous: CMOS *and* TFET	Dynamic	Dynamic	Hetero-Auto

Table 2: Evaluated multicore configurations. The performance of these schemes are presented in Section 6.

Parameter	Value
No. of Cores (Homogeneous)	32 CMOS *or* 32 TFET
No. of Cores (Heterogeneous)	8 CMOS *and* 24 TFET
L1 D/I-Cache	Private, 32KB each, 4-ways set-assoc., 1-cycle latency
L2 Cache	Shared, 4MB, 16-ways set-assoc. 10-cycles latency (2GHz)
Memory Access Latency	120-cycles (at 2GHz)
DVFS Epoch Size	1ms
Power Partitioning Epoch Size	5ms
Power Transfer Percentage	10%

Table 3: System parameters.

optimal, since a large chunk size can still result in load imbalance across threads and a small chunk size can significantly increase the overhead incurred during the request of a new chunk. A modified version of dynamic scheduling is *guided* scheduling [23], where the chunk size is not fixed, but variable. In guided scheduling, the chunk size starts large but gradually reduces as more loop iterations are completed, which achieves a smaller scheduling overhead. We modified our target benchmarks such that, all OpenMP parallelized loops use guided scheduling.

We implemented power partitioning as a part of our simulator. At every epoch, the total number of instructions executed by all cores in the processor is calculated and the total number of instructions per second (IPS) is obtained. This calculation can be implemented as an OS daemon process on a real system. In this case, an interrupt will be generated at every epoch, causing the daemon to read hardware performance counters and recalculate the total IPS in the last epoch. Comparing the total IPS for the current and previous epochs, it either decides to move power from the TFET domain to the CMOS domain or vice versa. The result of the power partitioner is a vector of per-core power budget values whose entries are stored in per-core power budget registers. Power re-distribution is carried at a period of 5 ms.

Our hardware DVFS implementation reads per-core power budget values from special purpose registers at every epoch. For each core, it determines the ratio of the power of the core to its power budget. It scales the voltage and frequency of the core such that it will match its allocated power budget in the next epoch. Note that in our heterogeneous CMOS-TFET multicore, DVFS is carried out on a per-core basis so as to take the difference in the power and performance characteristics of different types of cores into account. Assuming a 50 mV/ns latency for on-chip per-core voltage regulators as presented by Kim et al. [25] and a PLL locking time of a few microseconds, we set our DVFS epoch length to be 1 ms. By selecting such a large DVFS epoch length, we ensure that the performance overheads due to DVFS are negligible. Note that our baseline systems also employ per-core DVFS, in which case, the energy and area overheads of having per-core voltage domains is common to both our baseline homogeneous and proposed heterogeneous multicores. In our implementation, we assign DVFS levels at frequency intervals of 125 MHz, ranging from 500 MHz to 2 GHz.

5.3 Benchmarks

For the purpose of running our simulations, we use applications with train input sizes from the SPEC OMP 2001 suite [7]. These applications are parallelized using OpenMP [26] pragmas. We ran our simulations from the start until the end of each application, and measured the performance improvement based on the reduction in execution time of the application.

6. EXPERIMENTAL RESULTS

To evaluate the performance of our 32-core heterogeneous processor with 8 CMOS and 24 TFET cores, we consider a 32-core homogeneous CMOS and a 32-core homogeneous TFET processor as baselines. We assume a fixed power budget of 32W (i.e., 1W per core) in all evaluated configurations (we later analyze sensitivity to power budget). In all configurations, sequential regions of the target applications are executed on CMOS cores, except with the homogeneous TFET system where the sequential regions are also executed on TFET cores. During the execution of the applications, if any cores are detected to be idle, then the power budgets of the idle cores are redistributed among the active cores. For instance, only one core can be active during the sequential region of an application, in which case this core is allocated the entire power budget of 32W.

6.1 Hetero-Manual-Best vs. Baseline

We first compare the performance of our heterogeneous multicore against both homogeneous baseline processors, namely, a 32-core CMOS and a 32-core TFET multicore. Figure 11 shows the performance of our baseline and proposed configurations. We also include a *Best-Base* bar in this figure, which represents the maximum of *CMOS-Base* and *TFET-Base* performance, and normalize all bars with respect to *Best-Base*. The *Hetero-Manual-Best* scheme assumes that the application programmer maps parallel regions exclusively to either CMOS or TFET cores of the heterogeneous processor, whichever one achieves higher performance. We observe that, on average, our heterogeneous CMOS-TFET multicore performs within 4% of the *Best-Base* performance. In case of applications like *apsi* and *equake*, we see that *apsi* performs best on 32-TFET cores and *equake* performs best on 32-CMOS cores. In both cases, by using a heterogeneous processor (without any heterogeneity-aware dynamic schemes), we are able to reach the performance of the best homogeneous system, which shows that our heterogeneous multicore is suitable for serving a variety of applications.

6.2 Hetero-Manual-Best vs. Hetero-Manual-Worst

When comparing our heterogeneous multicore against the baseline systems, we assumed that the programmer is capa-

ble of making perfect mapping decisions. In other words, the programmer analyzes each application and decides whether it is better to run the parallel regions of the application on the CMOS or TFET part of the heterogeneous processor. Making a bad thread mapping decision (e.g., mapping threads to CMOS cores although TFETs would perform better) can lead to sub-optimal performance. In Figure 12, we compare the two programmer-directed mapping schemes Hetero-Manual-Best and Hetero-Manual-Worst, which represent the best and worst mapping schemes for the target applications. From this figure we observe that bad mapping decisions can degrade the heterogeneous system performance by 32% on average, which easily eliminates any benefit of using the heterogeneous system. Therefore, making correct mapping decisions on heterogeneous systems is key to optimizing performance.

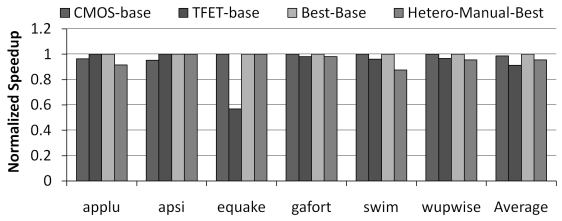

Figure 11: Performance of the baseline homogeneous and the proposed heterogeneous multicores.

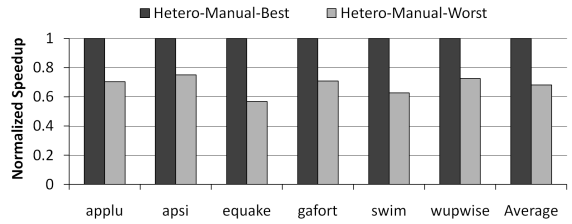

Figure 12: Performance comparison of the best and worst thread mappings on our CMOS-TFET heterogeneous multicore.

6.3 Hetero-Auto vs. Hetero-Manual-Best

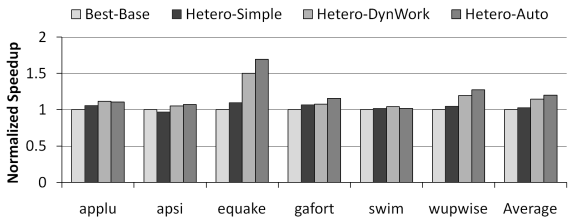

Figure 13: Performance of our CMOS-TFET multicore with and without dynamic work and power partitioning schemes.

We now evaluate our proposed dynamic power partitioning scheme and compare its performance against that of the best programmer-directed mapping. This dynamic scheme is performed *automatically* by a runtime system and requires no effort on the programmer's part. Note that, while the programmer-directed mapping uses only one type of core at any time, our dynamic scheme uses both types of cores simultaneously. Further, applications running under our scheme also use dynamic work partitioning, whereas the manual scheme assumes a static, equal work partitioning.

Figure 13 shows our results with the two schemes, as well as two intermediate schemes. The *Hetero-Simple* scheme uses the heterogeneous CMOS-TFET processor with only heterogeneous mapping, i.e., it employs neither dynamic work partitioning nor dynamic power partitioning. We can see that, heterogeneous mapping alone makes our heterogeneous multi-core perform 4% better than the baseline. Adding dynamic work partitioning on top of heterogeneous mapping (*Hetero-DynWork*) brings an additional 12% performance improvement. Finally, we observe that, on average, our dynamic power partitioning scheme (*Hetero-Auto*) performs 21% better than the best baseline performance.

In conclusion, superior sub-threshold characteristics of TFETs enable them to operate at points that cannot be realized by CMOS devices. Hence, by incorporating TFET devices, heterogeneous multicores can achieve higher performance, and serve a wider variety of applications. It is true that this improvement comes with an additional cost of using TFET technology and integrating CMOS and TFET technologies. However, a recent technology change to tri-gate transistors provided 18-37% improvements at the device-level [13], at a similar cost. Hence, we believe that the 21% system-level improvement presented in this work is very significant.

6.4 Runtime Characteristics

Figure 14 shows the dynamic variation in power consumption of *wupwise* application for the CMOS baseline (middle) and CMOS-TFET heterogeneous multicore (top), as well as the number of active cores (bottom). The total power consumption of both systems follow a similar trend: the entire power budget can be utilized in parallel regions, but sequential regions use only a fraction of available power as only a single core is active. The top figure also shows how our dynamic power partitioning scheme distributes the chip power across CMOS and TFET domains, which results in an improved overall performance.

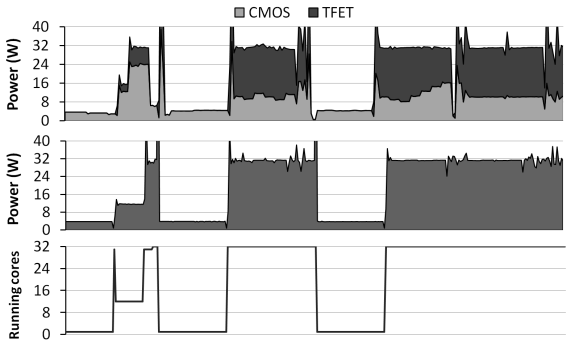

Figure 14: Power consumed by heterogeneous (top) and homogeneous (middle) systems, and number of active cores (bottom) over time (*wupwise*).

6.5 Sensitivity Analysis

As part of our sensitivity analysis, we show the performance of our heterogeneous processor under varying power budgets and ratios of CMOS and TFET cores.

• **Sensitivity to Power Budget:** Our results with the 8-CMOS, 24-TFET multicore with different power budgets for *wupwise* are shown in Figure 15. We observe that, for low power budgets, our heterogeneous multicore achieves a

higher speedup. The reason for this behavior is that when power is scarce, cores in the homogeneous CMOS system are forced to operate at very inefficient operating conditions when compared to TFET cores employed in our heterogeneous processor. On the other hand, the homogeneous TFET processor still suffers from the limited maximum frequency of TFETs, which significantly degrades sequential region performance. In contrast, the heterogeneous multicore is able to utilize the available power more efficiently, leading to significant speedup. Since future processor trends point to a decrease in available power per core, we believe that the performance improvement obtained by the heterogeneous CMOS-TFET multicore will increase even further.

• **Sensitivity to CMOS-TFET Core Ratio:** Figure 16 shows the variation in speedup with the number of CMOS cores in our 32 core processor. The two benchmarks demonstrated in this figure, namely, *wupwise* and *apsi*, show different trends as the CMOS core quantity is increased. *wupwise* scales well with number of cores and prefers running at low frequencies on the largest possible number of TFET cores. Yet, a single CMOS core is needed to achieve high performance while executing the sequential regions of the application. Therefore, neither the CMOS-only nor the TFET-only baseline performs satisfactorily and we achieve a performance improvement of up to 37% for the single CMOS core case. On the other hand, the scalability of *apsi* is limited, on account of which it prefers running on a moderate number of CMOS cores. In this case, the best performance improvement of around 10% is observed with the 4 and 8 CMOS core systems. Having more or less number of CMOS cores results in a shift in the operating point of all CMOS cores and leads to a relatively inefficient execution.

We also experimented with different power transfer percentages and found that while amounts higher than 10% lead to oscillations, smaller amounts (large enough to trigger DVFS level changes) still perform satisfactorily.

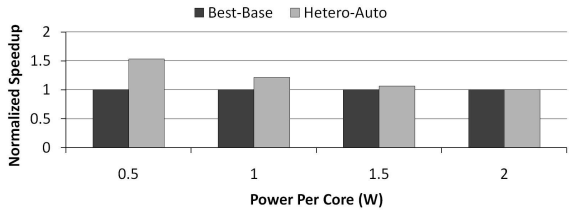

Figure 15: Performance of CMOS-TFET multicore under different power budgets(*wupwise*).

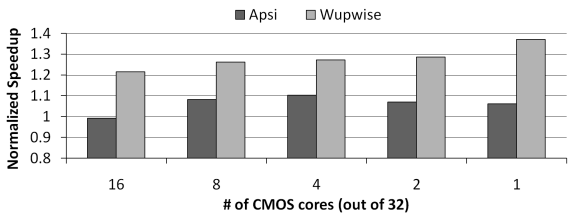

Figure 16: Performance of CMOS-TFET multicore for varying number of CMOS cores.

7. RELATED WORK

The ideas presented in this work can be distinguished from the prior work at two main points: constructing heterogeneous multicores and mapping applications to heterogeneous multicores. There have been various prior works on heterogeneous multicores, most of which consider architectural heterogeneity and process variations in multicore processors. [27] proposes a heterogeneous multicore processor to exploit different points in the power/performance space in an application's execution. [28] shows that heterogeneous multicores can bring significant power reduction for similar performance or significant performance improvement for the same power budget. [29] proposes a dynamically reconfigurable processor that can adapt itself to changing workload behavior dynamically. Showing that critical sections in applications can also be causes for serialization, [30] illustrates use of high performance core for critical section execution and low power worker cores for parallel execution. An analysis of heterogeneous computing using on-chip custom logic, reconfigurable logic, and GPGPU cores is performed in [31]. [32] uses specialized processors with high energy efficiency to fight against the power budget. Various prior works [33, 34, 35] focused on process variations, which is an unintentional source of heterogeneity in multicores. The common goal in these work is to address performance asymmetry across cores by finding the voltage and frequency levels that maximize performance.

There has been only a handful of prior work on device level heterogeneity. In [4], TFET devices are used to design processor cores that have very high efficiency at low frequencies. The authors show large improvements in energy efficiency of embedded processors by running sequential applications on TFET cores at lower frequencies. [36] presents a heterogeneous multicore processor running multithreaded applications under DVFS. They observe that, due to the dynamic behavior of applications and the unequal distribution of work, most of the execution time is spent at low frequencies. Exploiting the fact that TFET devices can provide significant energy savings at those frequencies, they propose a dynamic thread migration scheme across TFET and CMOS cores.

Most of the prior work in multicore heterogeneity is based on architectural heterogeneity and are orthogonal to what is presented in this work, and the works in device level heterogeneity used TFET devices only as a way to improve the energy-delay product of processors by reducing its power consumption, sometimes even sacrificing from performance. In contrast, in this work, our primary target is to achieve the *maximum possible performance while working under a maximum power budget*, which has not been studied in any of these works.

Various efforts have also been made to address the problem of how to map applications to multicore processors. In [37], the behavior of commercial applications running on asymmetric multicores is analyzed. The authors indicate that the application itself needs to be aware of asymmetry, and stress that when running multithreaded OpenMP applications on heterogeneous systems, guided scheduling typically performs better. [38] analyzes multithreaded applications running on a frequency-heterogeneous real multicore system and points out the use of hardware counters in quantifying application progress, which can in turn be used to determine the best number of threads to use at runtime. [39] also indicates the difficulty in mapping applications to cores on a heterogeneous processor and indicate that this is no longer a task that can easily be performed by the programmer. To simplify the process, they propose an automated method that performs runtime adaptive mapping of computations to processing elements. These prior works agree on the fact that simple static work partitioning constructs do not perform well for heterogeneous processors and some form of dynamic load balancing is required to better

make use of the performance asymmetry across cores. In an effort to efficiently port applications to heterogeneous processors, these prior work (i) run cores at their respective maximum frequencies without being concerned about power limitations (which required dynamic work partitioning due to performance imbalance) [27, 40], (ii) adjust the frequency/voltage of different types of cores so that static scheduling can still perform satisfactorily [33, 34, 35], or (iii) exploit architectural heterogeneity solely from a perspective that accelerates sequential or critical regions of applications [30, 29, 41]. None of the prior works consider accompanying dynamic work partitioning across application threads with *dynamic power management across cores* as a method to better utilize heterogeneous multicore processors, which is another contribution we make in this work.

8. CONCLUSIONS

ITRS projections for multicore chips indicate continuous decrease in per-core power budget which requires cores to be optimized to consume very low power. However, low power optimized homogeneous multicores cannot satisfy the needs of sequential applications, sequential parts of parallel applications, or applications with limited scalability. In such cases, heterogeneous multicores that use different types of cores for different purposes can perform better. However, it is not a straightforward task to efficiently port an application to a heterogeneous multicore. Extracting high performance from a heterogeneous processor requires both application-specific and processor-specific information and cannot be efficiently performed by the programmer. In this paper, we attacked the problem of how to extract the best performance from a device-level heterogeneous CMOS-TFET multicore. We proposed a runtime system that employs dynamic power partitioning to extract high performance from heterogeneous multicores without requiring any extra effort for application porting. Our power partitioning scheme works in conjunction with heterogeneous thread mapping and dynamic work partitioning, and distributes the available power across cores of the heterogeneous processor in an attempt to achieve better performance. Our results show that, on an average, under a fixed power budget of 1W per core, an 8-CMOS 24-TFET heterogeneous multicore with dynamic power partitioning can achieve 21% performance improvement over the best equivalent homogeneous multicore.

9. ACKNOWLEDGEMENTS

This work was supported in part by NSF awards 1147388, 0903432, 0903432, 1152479, 1017882, 0702617, 0963839, 1213052 and 1017882; and SRC NRI MIND. Karthik Swaminathan is supported by IBM PhD Fellowship.

10. REFERENCES

[1] S. Borkar. Thousand Core Chips: A Technology Perspective. In *DAC*, 2007.
[2] H. Wei et al. Scaling with Design Constraints: Predicting the Future of Big Chips. *IEEE Micro*, 2011.
[3] D.K Mohata et al. Demonstration of MOSFET-Like On-Current Performance in Arsenide/Antimonide Tunnel FETs with Staggered Hetero-junctions for 300mV Logic Applications. In *IEDM*, 2011.
[4] V. Saripalli et al. An Energy-Efficient Heterogeneous CMP based on Hybrid TFET-CMOS cores. In *DAC*, 2011.
[5] International Technology Roadmap for Semiconductors. 2011.
[6] B. Wheeler. Calxeda Spins 4W Server-on-a-Chip. In *Microprocessor Report*, Nov 2011.
[7] V. Aslot et al. SPECOMP: A New Benchmark Suite for Measuring Parallel Computer Performance. In *International Workshop on OpenMP Applications and Tools*, 2001.
[8] S. Mookerjea et al. Experimental Demonstration of 100nm Channel Length In0.53Ga0.47As-based Vertical Inter-band Tunnel Field Effect Transistors (TFETs) for Ultra Low-Power Logic and SRAM Applications. In *IEDM*, 2009.
[9] R. Gandhi et al. Vertical Si-Nanowire n -Type Tunneling FETs With Low Subthreshold Swing ($\leq$ 50mV/decade) at Room Temperature. *IEDM*, 2011.
[10] D.K. Mohata et al. Self-aligned Gate NanoPillar In0.53Ga0.47As Vertical Tunnel Transistor . In *DRC*, 2011.
[11] M. Luisier and G. Klimeck. Performance Comparisons of Tunneling Field-Effect Transistors Made of InSb, Carbon, and GaSb-InAs Broken Gap Heterostructures. In *IEDM*, 2009.
[12] U. E. Avci et al. Comparison of Performance, Switching Energy and Process Variations for the TFET and MOSFET in Logic. In *VLSIT*, 2011.
[13] Intel Corporation. Intel 22nm 3-D Tri-Gate Transistor Technology, May 2011.
[14] M. LaPedus. TSMC to make FinFETs in 450-mm fab, February 2011.
[15] C.C. Wu et al. High Performance 22/20nm FinFET CMOS Devices with Advanced high-K/metal Gate Scheme. In *IEDM*, 2010.
[16] Synopsys. *TCAD Sentaurus Device Manual, Release: C-2009.06*, 2009.
[17] Wei Zhao and Yu Cao. New Generation of Predictive Technology Model for Sub-45nm Design Exploration. In *ISQED*, 2006.
[18] Intel Corporation. Intel Atom Processor Z5xx Series - Datasheet, June 2010.
[19] Xilinx. Xilinx Power Tools Tutorial: Spartan and Virtex 6 FPGAs.
[20] A. Sinha and A.P. Chandrakasan. JouleTrack-a Web based tool for software energy profiling. In *DAC*, 2001.
[21] S. Datta et al. Ultrahigh-Speed 0.5 V Supply Voltage In0.7Ga0.3As Quantum-Well Transistors on Silicon Substrate. *IEEE Electron Device Letters*, 28(8):685 –687, Aug. 2007.
[22] V. Saripalli et al. Variation-tolerant Ultra Low-Power Heterojunction Tunnel FET SRAM Design. *Nanoscale Architectures*, 2011.
[23] C. D. Polychronopoulos and D. J. Kuck. Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. *IEEE Trans. Comput.*, 36(12), December 1987.
[24] P. S. Magnusson et al. Simics: A Full System Simulation Platform. *Computer*, 35, February 2002.
[25] W. Kim et al. System Level Analysis of Fast, Per-Core DVFS using On-Chip Switching Regulators. In *ISCA*, 2008.
[26] OpenMP. OpenMP, url = http://www.openmp.org.
[27] R. Kumar et al. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. In *MICRO*, 2003.
[28] T. Y. Morad et al. Performance, Power Efficiency and Scalability of Asymmetric Cluster Chip Multiprocessors. *IEEE Comput. Archit. Lett.*, 5, January 2006.
[29] E. Ipek et al. Core Fusion: Accommodating Software Diversity in Chip Multiprocessors. In *ISCA*, 2007.
[30] M. A. Suleman et al. Accelerating Critical Section Execution with Asymmetric Multi-core Architectures. In *ASPLOS*, 2009.
[31] E. S. Chung et al. Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs? In *MICRO*, 2010.
[32] Ganesh Venkatesh et al. Conservation Cores: Reducing the Energy of Mature Computations. In *ASPLOS*, 2010.
[33] E. Humenay et al. Impact of Process Variations on Multicore Performance Symmetry. In *DATE*, 2007.
[34] R. Teodorescu and J. Torrellas. Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors. In *ISCA*, 2008.
[35] U. R. Karpuzcu et al. The BubbleWrap many-core: popping cores for sequential acceleration. In *MICRO*, 2009.
[36] K. Swaminathan et al. Improving Energy Efficiency of Multi-Threaded Applications using Heterogeneous CMOS-TFET Multicores. In *ISLPED*, 2011.
[37] S. Balakrishnan et al. The Impact of Performance Asymmetry in Emerging Multicore Architectures. In *ISCA*, 2005.
[38] M. Bhadauria et al. Accomodating Diversity in CMPs with Heterogeneous Frequencies. In *HiPEAC*, 2009.
[39] C-K. Luk et al. Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping. In *MICRO*, 2009.
[40] R. Kumar et al. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. In *ISCA*, 2004.
[41] M. D. Hill and M. R. Marty. Amdahl's Law in the Multicore Era. *Computer*, 41, July 2008.