Communications to SIMAI Congress,
ISSN 1827-9015, Vol. 3 (2009) 267 (12pp)
DOI: 10.1685/CSC09267

Kernel- and CPU-level architectures for computing and A\V post-production environments

Walter Arrighetti[1,2]

[1] Technicolor *Creative Services,*
Digital Services for film and post-production,
via Tiburtina 1138, 00156 Rome, Italy

[2] *"Sapienza" Università di Roma,*
Department of Electronic Engineering,
via Eudossiana 18, 00184 Rome, Italy
riemann.chaos@gmail.com

Abstract

High-Performance Computing has been improving for the last decades through more parallelism and high-level machine instructions. Multimedia applications for Audio\Video post-production also rely on fast algebraic data manipulation, which is though not fully supported at CPU and OS kernel levels yet. After a brief review on current hardware and software implementations, several steering proposals towards future architectures for both HPC and A\V post-production environments, as well as OS human interfaces is sketched here.

Keywords: `High-Performance Computing, HPC, ISA, SSE, AVX, parallel architecture, mathKernel, Digital Intermediate, DI, A/V post-production, HDRI, grading, compositing, GUI, human interface, userspace, pipe`

1. Introduction

Bare CPU arithmetic power for High-Performance Computing (HPC) has been improving for the last decades both vertically and horizontally through higher- and higher-level instructions sets (ISAs). Verticality refers here to the increase of CPU clock speed, as well as system bus and network infractructure bandwidths. Horizontality refers to the massive increase in parallelism at different levels: multi-core CPUs with longer pipelines, multi-processor machines, up to cluster and GRID computing relying on more computing nodes cooperating together to the same task (either connected in LANs, WANs or across the Internet).
Multimedia applications for consumer, as well as for professional

Received 13/02/2009, in final form 15/06/2009
Published 31/07/2009

Audio\Video post-production facilities, also rely on fast and massive algebraic data manipulation which is though, except for a few cases, not fully top-down supported throughout the complete computing workflow (i.e. hardware to software).
Times are now mature for a new paralellized computation paradigm which relies on both the processor's ISA and on the Operative System (OS) kernel to provide services and applications (API-driven or not) with a unified HPC "platform" [1].
This consideration is simply driven by the fact that common PC applications run, at low level, similar (if not identical) operations which scientific computation demands — that is, more or less, numerical Linear Algebra, which is almost everything we can actually compute.
As a matter of fact, applications with almost any purpose —from any graphical user interface (GUI) to multimedia, from office to graphic design suites, from Internet browsers to videogames— perform bilions of integer or floating-point operations per second (FLOPs) on a regular basis which, stripped off their high-level informative content, turns down to be either a boolean search (like in databases), or the direct or inverse solution process of a system of linear vector equations like

$$A\mathbf{x} = \mathbf{b}$$

or its matrix analogue, $AX = B$. Deepening a bit more in the inside, let us consider a multi-media application like a video player as an example.
At high-level, A\V streaming consists in uncompressing and decoding a binary file (usually real-time) and representing its raw stream as sound and colour pixels on-screen. At the OS level, that is usually performed by parallel-tasked threads decoding different pieces of the same binary data at the same time (I\O access times apart). At low-level, there is the CPU issuing SIMD (single instruction, multiple data) opcodes that perform many instances of string substitution and discrete cosine transform (DCT), accordingly. In a cluster environment (like a video post-production facility), the scenario has another highest level, where one master node delivers pieces of multimedia computation data to several computing nodes, each simultaneously processing different parts of the same data, as happens for render-farms like that depicted in Fig. 1.

Abstracting away from the software/hardware layers (and the machine code underneath), the logic level of such computations is, as previously said, nothing more than finite-precision Boole's and Linear Algebra: machine instructions basically do nothing more than shift, rotate, compare and perform boolean and basic arithmetic operations on binady data stored inside the CPU's registers.

DOI: 10.1685/CSC09264

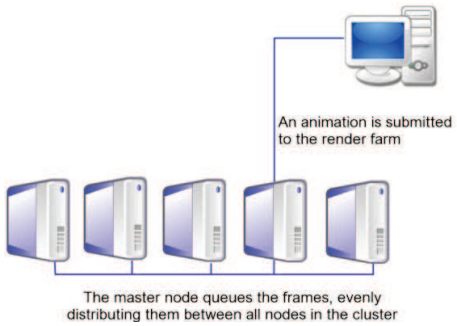

Fig. 1. Parallel multimedia computing paradigm of a render-farm.

2. Current CPU architectures

In the last decade, it has become clear that simply increasing CPU clock speed is not a good strategy to raw computing performance: with present microelectronic processes, raising the clock above 4GHz is a "no-go" due to the overwhelming thermal drift produced by transistor gates' commutations. In order to counteract for that, more powerful cooling is to be applied to the microprocessors themselves, requiring expensive and encumbring cooling. Currently, a steering toward parallelization is carefully pursued, where several computing cores are hosted on the same die.

A first strategy is pure quantity: multi-processor computers are becoming the standard in office and professional facilities, whereas each CPU features inside whole replicæ of its core μ-architecture. Another strategy, as depicted in Fig. 2, is employing more and longer pipelines: for each issued instruction there are others already queuing up or even being partially processed and executed (farther along the pipeline). As a side-effect internal cache memories grow larger, and higher-level caches get integrated more and more inside CPU cores (with Intel introducing, in late 2008, the Core^{TM} i7 processor line, featuring RAM boards directly connected to the CPU).
A third strategy is to parallelize some of each pipeline's stages: the Arithmetic-Logic (ALU) and the Floating-Point Unit (FPU). On a n-bit CISC μ-architecture these components have some tens of *packable* registers, which store either one maximum-width value, or more than one with decreased width: for example, a 128-bits register stores either one *oct*-word, two quad-words (64 bits each), four double-words (32 bits), eight words (16 bits) or sixteen bytes (8 bits) — cfr. Fig. 3.

As far as the mathematical μ-architecture is concerned, current manufacturers usually deploy a *continuation* of their CPUs' ISA for backward compatibility's sake: when a new family of processors is introduced, their

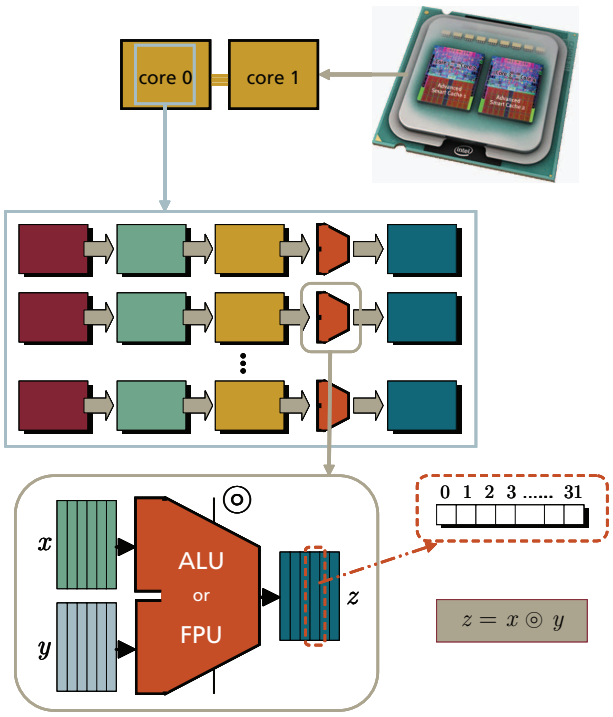

Fig. 2. From multi-core to pipeline, to ALU/FPU microarchitectures.

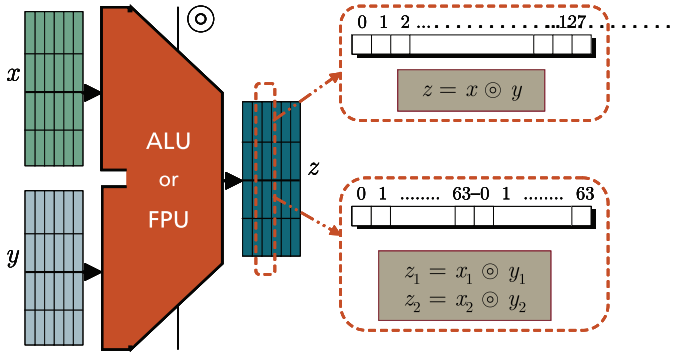

Fig. 3. Packed registers in a ALU or FPU.

common ISA is a super-set of the previous one, where new instructions provide additional functionality (although their μ-electronic implementations are often complete re-engineerings rather than add-ons to existing ones)

DOI: 10.1685/CSC09264

3. Mathematical-oriented instruction sets

Referring to Fig. 4, Intel's ISA continuations for integer and floating-point (FP) instructions quickly steered from x87 16-bits math co-processors to MMX first, then to SSE*n* extensions (currently at their 4^{th} version), featuring extensive parallel Boolean and arithmetic SIMD instructions on packed registers, plus FP arithmetics and lots of opcodes, for all-in-one dot-products and comparisons, [2]. Currently under development, Intel's Advanced Vector Extensions (AVX) and AMD's SSE5 provide new enhancements to SIMD computing. Despite being used on in 64-bit architectures, both will feature new 128- and 256-bits wide packable registers, 3- and 4-operand instructions (like fused products or conditional/predicating branches) and more high-level opcodes: from FP arithmetics, powers and square roots, comparisons and max/min evaluations, up to optimized codes for AES data encryption. Furthermore, the new super-sets are optimized for internal parallelism of multi-core CPUs (quad- and up), [3].

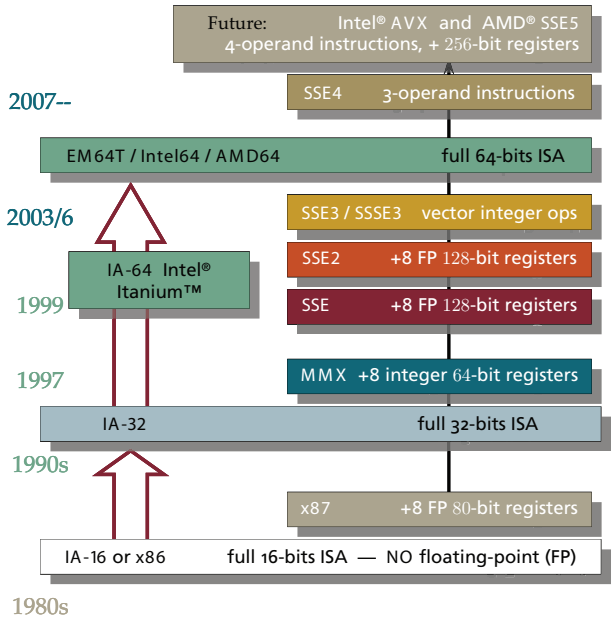

Fig. 4. Timeline of Intel/AMD mathematical/multimedia-oriented ISAs.

As depicted in Fig. 5, among the most powerful SIMD operations, there are those specifically targeted to Linear Algebra, i.e. mainly dot and matrix products. Since they can be combined among different levels of multidimensional arrays and, at the same time, the distributivity of the arithmetic operations performed is also programmable too (both in low- and high-

level languages), these are prototypes for more general inner and outer (tensor) products. In Computer Science instead, such are usually referred to as *fused multiply* operations: given two ALU\FPU operations $\oplus$ and $\odot$ and three suitably-sized arrays (tensors) $\underline{X}$, $\underline{Y}$ and $\underline{C}$, a fused operation is something like $\underline{Z} = \underline{X} \odot \underline{C} \oplus \underline{Y}$ (with given algebraic precedence rules). In the case of $N \times N$ matrix products, the above expression $Z = XC + Y$ is computed, coordinate-wise for $1 \leq m, n \leq N$, as:

$$z_{m,n} = \sum_{h,k=1}^{N} c_{m,k} x_{h,n} + y_{m,n}.$$

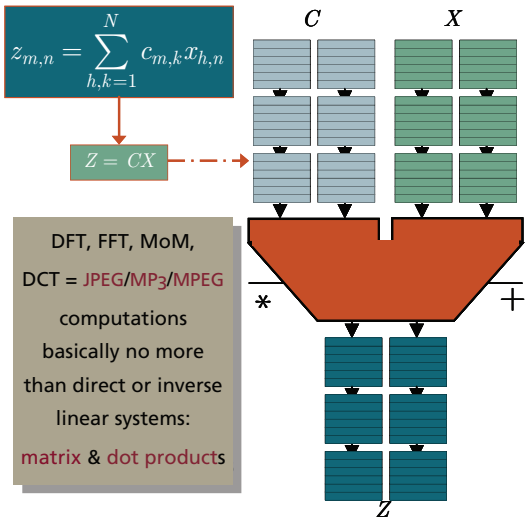

Fig. 5. Example of SIMD outer product on a FPU.

Dot/outer products (or fused multiplications when they are parallel-performed by a machine) are the basic mathematical operations hugely performed by many general-purpose software, from multimedia codecs and streamers (like the Fourier transforms needed by JPEG/MP3/MPEG formats or DSP in general) to videogames, 3D render-farms, motion picture post-production facilities, as well as HPC of course.

4. A digital imaging example

The way such kinds of massive computing capabilities are (or should be) exploited in the multimedia industry (and above all that of digital imaging and cinema) can be easily seen in the example of Fig. 6, where the picture is a stack of several transparent layers whose every pixel is an active mapping

DOI: 10.1685/CSC09264

of the overlayed pixels beneath. The image's final dynamic range (transition from shadows to highlights —i.e. from over- to underexposure) and colour look can get dramatically changed. This process, called *colour grading* (or -correction), is obtained applying linear or nonlinear maps to each of the pixel's image, for each of its colour channels. These are typically red, green and blue —hence the RGB colour-space— but others are often used, like YUV-derived ones for most of the analogic/HD video world, or the CIE XYZ for Digital Cinema. Whenever images are transferred between media (for example from a film scanner or a digital camera sensor to a post-production software, whence to either monitor, printed paper, or other film stock), conversions from the source colour-space into either a variant of it or into a different one are needed. For discrete spaces, those are usually performed via a colour look-up table (LUT), often in the shape of a *colour cube* (aka 3D *LUT*), which is a graphical representation of mapping some sample input colours into a corresponding output palette, whereas intermediate colours usually undergo to some aliasing like trilinear filtering — all for colour reproducibility's sake.

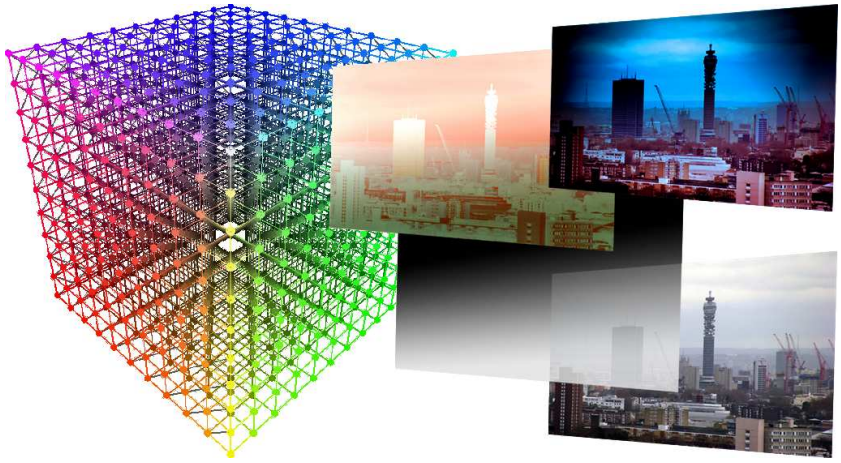

Fig. 6. A 3D LUT (left) and layer-based correction for still-image colour-grading (right).

Let $(\mathbf{R},\mathbf{G},\mathbf{B}) \in [0,1]^{3N^2}$ be the block-vector representing the original pixels (sparated by colour channels) of a $N\times N$ image; colour-grading as a layer uniformly mixing the colour channels of the global image is a block-orthogonal matrix (in this case not depending on nearby pixel values), which maps every pixel colour to a new one, $(\mathbf{R}',\mathbf{G}',\mathbf{B}')$, as:

$$\begin{pmatrix}\mathbf{R}'\\ \mathbf{G}'\\ \mathbf{B}'\end{pmatrix} = \begin{pmatrix} c_{\mathsf{R}} I_N & C_{\mathsf{RG}} & C_{\mathsf{RB}} \\ C_{\mathsf{GR}} & c_{\mathsf{G}} I_N & C_{\mathsf{GB}} \\ C_{\mathsf{BR}} & C_{\mathsf{BG}} & c_{\mathsf{B}} I_N \end{pmatrix} \begin{pmatrix}\mathbf{R}\\ \mathbf{G}\\ \mathbf{B}\end{pmatrix},$$

where $c_{\mathsf{R}}, c_{\mathsf{G}}, c_{\mathsf{B}} \in \mathbb{R}_+$ are constants for occasional shifting the three pure colours and I_N is the $N \times N$ identity matrix.
In practice this operation, if performed on a camera- or scanner-generated picture, corresponds to millions of similar fused multiplications on each of the image pixels (channel by channel): a perfect meal for nowadays number-crunching CPUs. Even nonlinear examples (like photo-realistic filters or visual effects) do not bias too much from that simple paradigm (as far as the mathematical model is concerned) — despite many of them are applied to the some discrete linear transforms (like Fourier's, or the wavelet one employed in D-Cinema's JPEG2000 and the RED$^{\text{TM}}$ CAMERA formats). Last but not least, nowadays digital pictures "live" in a *high dynamic range* (HDR) 3-dimensional colour-space, i.e. with integer or "float", 10- to 32-bits-*per*-channel, in order to keep more information on both under- and over-exposed regions. That is where CPU registers' width and their FP capabilities are becoming more and more of a turning point.

5. Motion picture post-production and the Digital Intermediate

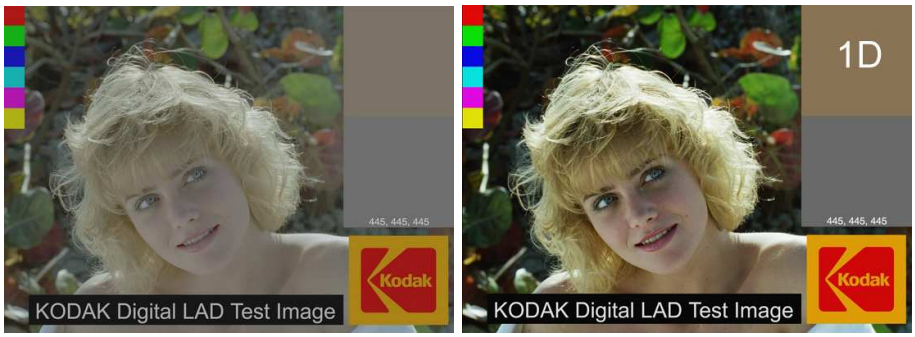

Fig. 7. First step in a DI workflow: logarithmic-to-linear colour-space conversion, in company of the beautiful Marcy (source scanned from internegative film).

Adding motion to high-quality HDR images —currently up to "8K" resolution, i.e. 8192×6226 pixels— (like the DPX file sequences produced by a film scanner and shown at 24fps) leads to the heart of the cinema industry's Digital Intermediate (DI) workflow, where HPC is essential to deliver, because colour grading and the other rendering processes involved (from transcoding to visual effects, to pure 3D, to downsampling) need to be performed real-time, still taking weeks to complete for a full-length movie, [4]-LustreCM. After being either scanned from film stock, or directly transcoded from a digital videocamera (like Arri's or RED$^{\text{TM}}$'s), frame-image sequences need to be digitally cleaned (to reduce chromatic noise and film imperfections)

DOI: 10.1685/CSC09264

and then conformed (via edit-decision lists, or EDLs). Colour-space conversion of digital images, from logarithmic ones (preserving film density information) to linear ones, are massively parallelizable operations, preceding the actual colour grading process. The latter involves Linear Algebra operations too, eigher globally (primary grading) and locally (secondary grading) performed at individual frames or whole clips, cfr. §4 and Figs. 7-8. In the end, the whole clips are prepared for other visual effects (like 3D graphics in some cases) and the work is finally rendered (and optionally downsampled to a D-Cinema package file for digital delivery), or back "lin-to-log" colour-space converted for film-out printing.

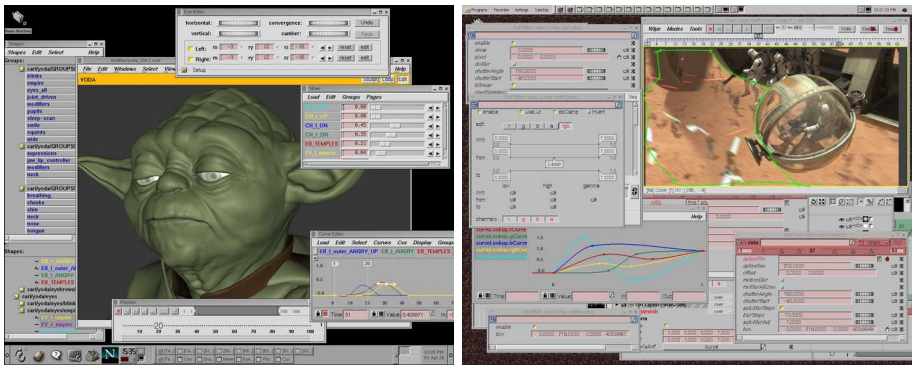

Fig. 8. Next steps in the DI: 3D sculpturing of a puppet to be renderd (left) and compositing of 3D models over a real scene (right). Courtesy of Lucasfilm.

6. Proposal for math-oriented microarchitectures

In order to further improve computational requirements of both scientific and non-scientific facilities (like those specialized in graphics, multimedia, and D-Cinema for example), future CPU microarchitectures should seriously consider providing more targeted opcodes for mathematical instructions.

Apart from improving parallelism in SIMD and MIMD opcodes (for massive matrix and dot product operations) as well as widening internal cache memories, support for two-dimensional registers and inclusion of cryptographic and expression-matching opcodes is required to strenghten databases and data-mining applications, as well as providing faster and more secure cryptography: generation and comparison of keys and hashes all within the CPU and its caches leverages overall security, at least for smaller data chunks.

Another feature that scientific computing would directly benefit from is wider IEEE-754 floating-point packable registers (512-bits and up), improved native support for opcode-level complex-numbers arithmetics and

basic nonlinear functions: integer powers, roots' and logarithms' (principal) branches, trigonometric and hyperbolic evaluation, cartesian-to-polar conversion, and so on. Availability of hardware-coded mathematical constants would also be a plus. Complex-number arithmetics and exponentials, in particular, require little variations in the internal logic circuitry, yet providing many improvements even in non-scientifical applications (like the DFT and FFT deeply used in multimedia). Despite many low- and high-level languages support complex numbers with few emulation efforts by the CPU, implementing them at the ISA level would ease many $(\cos z, \sin z)$-like mappings off the compiled code. The ideal situation is being able to issue an iterated assembly-code sequence like

```
cMUL(cINV(cPOW(PI,IMAG)),cSQRT(cDIV(cADD(1,-cPOW(2,cSINH(XAX)
  )),cLOGE(cLOGE(ADD(CONJ(XAX),-cMUL(IMAG,cSQRT(5)))))))
```

to compute (at least the principal branches of) expressions like:

$$\frac{1}{\pi^{\imath}}\sqrt{\frac{1-\sinh^2 z}{\log\log\left(\bar{z}-\imath\sqrt{5}\right)}} = (-1.53280\mathrm{E}-3, .53281\mathrm{E}-3).$$

In many consumer to high-end equipments there is currently a trend to exploit the massive computing power of a Graphics Processive Units (GPUs), whose core market is that of 3D videogames. Computingwise, a GPU is a massively-parallel microprocessor (now featuring fastest and dedicated own RAM, wide buses, registers and high clock speeds). Despite GPUs of the past having specialized internal logics, making them unsuitable for generic-purpose HPC, recent ones do not: e.g. *n*VIDIA manufactures multicore 256-bits GPUs, also motherboard-integrated. Many scientific applications, even large-scale GRID projects like Folding@home, employ techniques borrowed from algorithmic computer graphics to benefit from their characteristics. Professional graphic adapters are (and will be more and more) engineered to manipulate large-sized HDR images (which means wide-register support for massive integer and FP number-crunching), whereas their *pixel-* and *vertex-shader*s sub-circuitries are completely *pre*-programmable, which means that tiny bits of machine-codes can be ingested within their caches and serially issued by single opcodes for fastest execution: that's exactly in a mathematical-oriented CPU's needs to run critically fast looped algorithms.

7. Proposal for a tentative mathKernel architecture

In the last section, conclusions are drawn regarding another critical issue: a OS kernel able to bridge the gap between hardware and software, which is not only the way to optimize HPC from high level, but also to ease

DOI: 10.1685/CSC09264

or even void architecture transitions for the developer's (and, ultimately, the user's) point of view. As depicted in Fig. 9 a typical kernel, as Linux's, features a low-level virtual memory environment —the *kernel-space*— where basic hardware I\O, thread and memory management take place, and the *userspace*, providing common frameworks for applications to run and rely on (like, ultimately, dæmons and standard libraries).

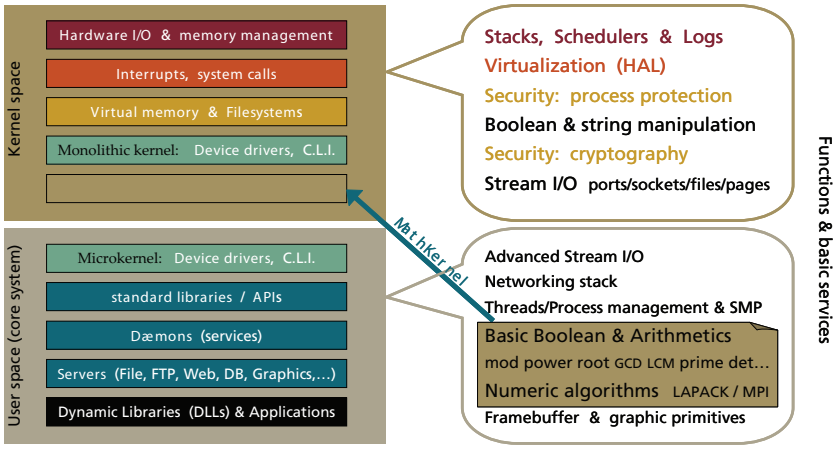

Fig. 9. Placement of the mathKernel in the simplified paradigm of a OS/kernel architecture, depicting just a bunch of its major features.

What is here referred to as *mathKernel* is a kernel module, loaded at boot time, providing low-level mathematical techniques to parallelize computations (even when issued by separate threads) as well as high-level mathematical functions. As stated at the beginning, many common applications (from multimedia, to Internet, to graphic design, to videogames) heavily rely on mathematical capabilities to run but, in fact, each one provides its own set of mathematical libraries, usually running in the userspace. A single framework of mathematical functions, common to every applications has, instead, many advantages. First of all, its source code can be improved, recompiled, relinked into the kernel and be immediately available for every applications to benefit (so no individual application update is needed when just the mathematical code is). Secondly, running the same set of mathematical functions (although each application high-level tailors it to its own needs and specific —even proprietary— algorithms) means that they can be better managed and parallelized in kernel-space. Executing different-tasked threads in parallel is, in fact, uncommon even for modern OSs, because underlying code coming from different services/libraries in userspace, is usually unhomogeneous and hardly parallelizable.
As a last proposal, the *mathKernel* could be integrated at the Linux

filesystem level too: a series of *pipe* devices are set up for different algorithms such that piping MathML expressions into them activates the corresponding kernel-space algorithm streaming out the results. For example, let `/dev/math` be the *mathKernel*'s device folder, containing pipes to particular mathematical functions, procedures, algorithms or methods, like `det`, `linsolve`, `GCD`, `ODEsolve`, `DFT`, `integrate`, `factor`, etc. Two `bash` commands like those below send a discrete time-domain signal stored in a XML file (which may also contain metadata parameters about the DFT algorithm to apply) to the Discrete Fourier Transform pipe (file `/dev/math/DFT`) and retrieve "on the background" its DFT, outputting to another file:

```
cat time-domain.xml > /dev/math/DFT &
(cat </dev/math/DFT > spectral-domain.xml &&
     echo -n "The mathKernel computed a DFT.") &
```

Integration scenarios in a Linux GUI like `xgl` are countless: examples include drag-n-dropping a Wolfram *Mathematica* notebook including already-graphed plots over specific algorithm pipe icons, automatically popping up the output graph whenever ready, with all mathematical computations optimized for the specific ISA, transparently parallelized in kernel-space and rendered in a new graph by the *Mathematica* front-end. Consider also dropping any files over the corresponding cryptographic-pipe icon to retrieve their AES-encrypted versions out from a specific pipe-folder.
High-level services might also be available regarding motion-picture industry (as sketched in §5). A full DI movie, composed by frame-file sequences (as DPXs), multi-channel audio and XML files for EDLs, grading/compositing projects, subtitles and a 3D film LUT, would be converted into a D-Cinema package (one DCP file) by simply drag-n-dropping its overall-project folder onto the corresponding high-level conforming icon: all the workflow would be automatically started in kernel-space.

REFERENCES

1. A. S. Tanenbaum, *Modern Operative Systems*, 3rd ed., Pearson, 2007.
2. Intel's Technology website, www.intel.com/technology/.
3. *Intel AVX Programming Reference*, Intel's software network, sofwareprojects.intel.com/avx/.
4. J. James, *Digital Intermediates for Film and Video*, Focal Press, 2005.
5. Autodesk® Color Management, autodesk.com/lustre-documentation.