IntellaSys Corporation
10080 N. Wolfe Road, Suite SW3-190
Cupertino, CA 95014
voice 408-446-4222
fax 408-446-5444

Defining Processing Solutions for Mesh Computing Environments

David Guzeman, Chief Marketing Officer,
IntellaSys Corporation, Cupertino, CA

Over the years the concept of mesh computing networks – the so-called sea of processors – has held a fascination for computer scientists and silicon jockeys alike. Everywhere you look, there are potential applications that lend themselves well to a computing environment that consists of anywhere from a handful to thousands (or even millions) of small processing elements. In this paper, we will consider a few of these applications and the implications they have on the computing elements that drive them.

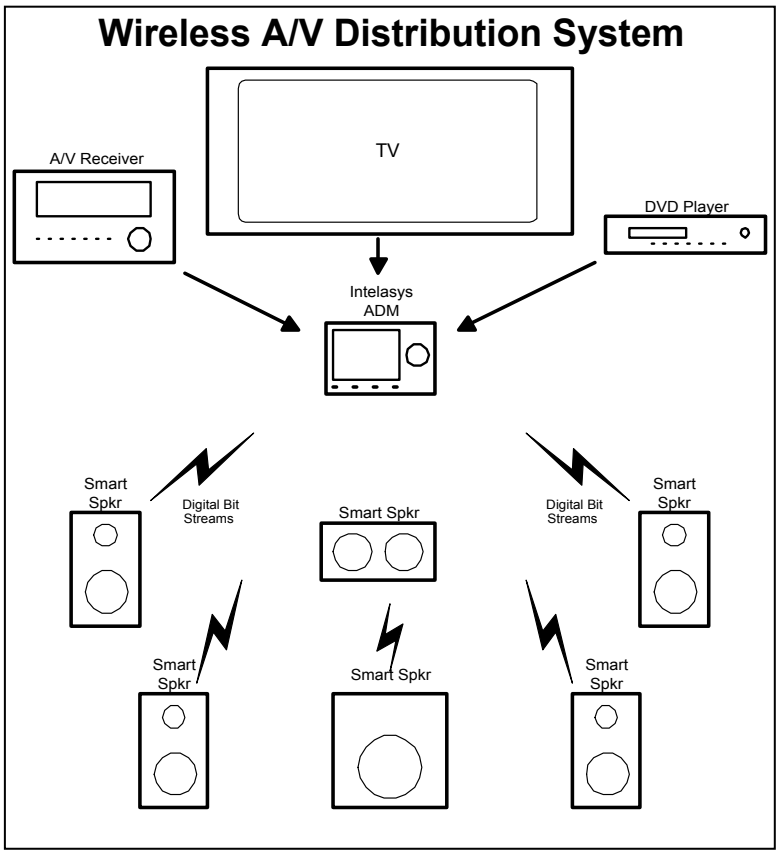
Sample Applications

When considering mesh computing networks, it is important to recognize that there is a tremendous scale of applications being served, both in terms of the complexity of processing that must be done at each node, and in the total number of nodes involved. Following are a select number of applications that together provide insight into creating the ideal processing solution for mesh computing environments.

Wireless Home Theater Systems

A mesh computing environment need not be large to deliver significant benefits. One of the smallest examples of mesh computing is one that would not normally even be thought of as a mesh network, namely the home theater system utilizing wireless speakers. Wireless speakers are just beginning to appear on the market but are currently limited to the "rear" effects speakers where cables tend to be the most burdensome to route and install. However, there is no reason why all of the speakers cannot be handled wirelessly wherein the number of remote nodes is at least six and may be more. In the case under consideration, audio information is transmitted digitally over a wireless link from the Audio/Video (A/V) receiver to the six speakers which decode the audio signal and drives the powered speakers.

Properly designed, the same computer chip can be used both at the A/V receiver to encode the audio into 5.1 surround sound format and to digitize it and transmit wirelessly AND at each of the speakers to receive, convert to digital, decode it, and convert it back to analog to drive the powered speaker. Moreover, all the nodes can be made bi-directional, meaning the speakers can also transmit back to the A/V receiver. This would allow all sorts of auto-calibration to take place as well as other, complex signal processing. In this case



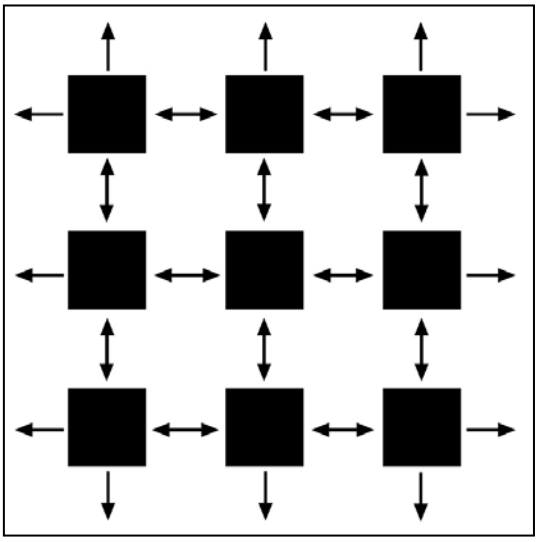
then, the mesh network would consist of seven nodes, one at the A/V receiver and six at the speakers. All would be symmetric in the sense that they share most or all of the same hardware. Likewise the speaker nodes would be identical in that they all share exactly the same computer code. While all speakers would receive the entire bit stream, each speaker would simply extract the audio information for that specific speaker.

Here we see one of the common elements to this class of mesh network, i.e., many of the nodes are totally identical and differ only in physical placement or geographical location. Others function as "servers" in that they provide data to the mesh and extract information back out of it. The placement of the nodes on the mesh is somewhat arbitrary. There is no logical difference, for instance, between the left and right speakers, other than they have identified themselves as left and right and are extracting the audio stream for the left and right respectively. The operator could push a button on the A/V receiver and switch left and right without moving or reconnecting any cables.

At the same time, this system differs in one way from what we would normally think of as a mesh – the highly defined routes that all emanate from one node and connect the others. It wouldn't make much sense, for instance, for the A/V receiver to send all of the data to just the subwoofer and have it then pass on data to the others. We have included the home theater system as an intriguing example of the unexpected places these networks can turn up when you're given inexpensive and very flexible node computing elements.

Sensor Driven Networks

There are many applications consisting of various types of sensors connected to local computing elements that are then interconnected in a mesh that serves to monitor and collect data from the sensors. On a small scale, a home security network fits this description. On a larger scale there are many scientific studies monitoring wildlife, ecological conditions, weather, or even earthquake fault zones. Today solid state sensors come in a wide variety that can monitor many types of phenomena, from the presence of specific gases to



accelerometers measuring and detecting motion, as well as the obvious ones for tracking temperature, light, humidity, etc.

Once again the placement of the nodes on the mesh may be very arbitrary. In the case of the home security system there is no need to know the mesh route to any particular sensor as long as that sensor provides an ID as it sends along data. In the case of the earthquake sensors, of course one arrangement would have the sensors placed in predetermined spots on the mesh, while another solution would be to let the nodes acquire and transmit their GPS coordinates along with their data. This second configuration is extremely flexible and has the advantage that it is much easier to set up and maintain. It is almost always critical that the mesh know the location of the nodes so that as data is acquired by those nodes it can be organized and acted on. That location can initially be determined by either setting it into the node (as in the case of a thumbwheel switch) or having the node identify itself in some way.

One of the other characteristics of this class of mesh networks is that nodes should be able to be added or removed casually without having to reconfigure or reprogram the system. Whereas in the home theater environment, new speakers are added very infrequently, in a mesh of scientific sensors or even in the home security network, adding new nodes should be trivial and easily done by untrained personnel.

Traffic Light Controllers

Who has not cursed the absurdity of the American traffic light system extant in most cities where each light operates independently of all others, oblivious to the traffic conditions at neighboring intersections to say nothing of the obvious traffic conditions at the prior intersection? To state the obvious: "Why am I stopped at a red light when there is absolutely no traffic coming the other way?"

Most people, when contemplating this system, picture the incredible improvement that could be brought to bear by a central computer located somewhere "downtown," monitoring the conditions at all of the intersections and by some clever algorithm controlling the traffic lights to maximize traffic flow and minimize our blood pressure. But as delicious as this fantasy is, consider that nothing as complex as a system "downtown" is needed.

What is actually needed is a modicum of intelligence at the traffic signals themselves. First of all, being aware of the conditions at the intersection would make a staggering improvement for most of us fighting our way through city traffic. But extending this to include knowledge of what's going on at neighboring intersections greatly improves the ability to handle traffic even more. Indeed, what is needed is not *central* intelligence, but a little bit of

distributed intelligence. Sounds like a job for mesh computing.

Traffic light control is actually a compelling example of a mesh network. Hundreds of nodes, each doing some local processing while talking to the nodes at neighboring intersections. And the computing element does not need a magical algorithm to create the perfect traffic flow, whatever that is anyway. It only needs to improve it a bit to make a big perceived difference. If the node knew when its neighbor was changing the signal and sending traffic its way, and what speed limit and distance was, it automatically knows when it will be getting a new batch of cars to process. Even simpler, at the time the system is installed someone could simply get in a car and drive the distance to that neighboring intersection and plug that parameter in.

Notice that we're making no attempt to understand the traffic conditions across the entire city, just the conditions at the neighboring intersections, a much simpler task. That also means there's no need to string cables across the city either. Each node only has to be connected to the neighboring nodes, and for that, wireless is fine. If you're concerned that interference on the wireless link might cause serious problems, even accidents, remember that all we need is to be sure the link data is valid. If not, we can always resort to just analyzing the conditions at our intersection and that alone would be a giant improvement over what we have now.

And since this is about improving the system, let's see what else we can do to smooth traffic in our cities. First, there is a new fad in our major cities of adding video cameras to catch people running red lights. This effort has been driven by accidents in the intersections with people running the light as it changes, and the resolution has to be sufficient to read license plate numbers. Why not control the camera with the node computer to eliminate the need to save video of empty intersections. The computer could certainly analyze whether a car is actually *in* the intersection during the transition period, and if not, discard the video. What if there IS a car there? Then video could be sent to the adjoining node to see if the same car speeds through the next intersection as well. In this way you get a record of the bad offenders and hopefully you do something about it.

Of course with a camera operating at the intersections you get all sorts of data. You know how many cars are passing at what times, how many are in which lanes, how many turn, etc. – a complete profile of the traffic at that intersection every day. And each night the data is rippled across the city, from signal to signal, until it arrives at some central collection point – perhaps the *uber lighten* in front of city hall.

To Catch a Terrorist

Imagine a computing node on the mesh consisting of a powerful computing element, an accelerometer sensor for detecting motion (footsteps), a GPS geoposition chipset to determine the exact position, and perhaps even audio and video sensor/cameras. Now let them communicate wirelessly in a mesh network. Could you build them for $1,000 each? In a heartbeat! More like $100 or less. But stick with the $1,000 figure a minute. Make a million of them! Now fly over Torra Borra and dump them from an aircraft, much the same way the US Navy dropped

sonobuoys from a P3 Orion to detect submarines. Imagine a million nodes on a mesh network that knows, through the GPS, where the nodes are, and can organize itself, that can talk from node to node, and pass data, images, and audio across the mesh, back to a control point. Total cost = $1 billion. A lot of money but it would tell you about every living thing that crosses the area, man or animal, and send images back of the transient. One thing that is common to all mesh networks is the way the nodes talk among themselves. Here's an imaginary conversation:

"Tell the humans the battery on node 54321 is running low and really should be replaced"

"I hear something. Sounds human"

"Good gosh, it's a tall dude in a white robe… here's some images of him… pass the word along. Node 1583 – he's moving your way, pick it up"

Fanciful? Sure, but practical? Yes!

Mesh Node Characteristics

Low Node Cost

As the number of nodes in the mesh increases, it is generally important that the cost per node is relatively low. Obviously users want the per node cost to be as low as possible, but in mesh networks the practicality of the mesh solution is frequently dictated by the node cost. This implies very small, inexpensive chips that are highly integrated and require little in the way of supporting silicon chips to complete the node.

Node Independence

A second mesh node characteristic is the high degree of independence each individual node has. Many mesh networks have intermittent and infrequent data transmissions from node to node, so that for the majority of the time they are working on their own. Additionally, reliability of the entire system dictates that the nodes *continue to operate even in the absence of communication with the other nodes.* Thus if the network route breaks down, the show goes on! There are two corollaries of this node independence:

1. Absence of Central Operating System

If the nodes are to be truly independent, it means there cannot be a traditional central operating system. Nodes might be directed to enter a specific mode or go to a particular state in a state machine, but once that's done they should continue to operate in that mode or state until directed otherwise.

The absence of a tightly-coupled operating system also means that you cannot count on all of the nodes being in the same state at the same time – some will simply get the word later than others, hopefully in a well designed mesh, close to the same time but not instantaneously.

2. High Computing Power at each Node

With node independence, each node needs to handle its own computing needs locally. In the case of sensor driven networks, the nodes frequently need to process and filter a continuing stream of data from the sensor without passing on every data bit to the network.

In some applications, the system is only practical IF the nodes are staggeringly fast computers. In something as simple

as wireless home theater speakers, the node at the A/V receiver has to do a full 5.1 surround sound encode and deliver that as a bit-stream to the speakers over wireless links, which it is actually implementing at the same time. And the computing elements at the speaker nodes have to be able to monitor that bit-stream and extract the appropriate data for their specific speaker. Ideally, these node computers would also perform the A/D and D/A conversions digitally as part of their programmed tasks.

Consider the case of a node that must monitor and process the output of an accelerometer to determine, for instance, if the vibrations it's picking up are caused by human presence. At the same time it's processing data from the accelerometer, it may be required to service a CCD camera encoding the image into a JPEG format, at the same time it is recording sound and compressing that data into an MP3 format, at the same time it is monitoring data on the mesh network, and in this case, sending its own data stream of vibrations, MP3 and JPEG multimedia, out onto the mesh. Such tasks are beyond the ability of conventional microprocessors, which is why it makes sense to pack dozens of high-speed core processors onto a single chip, each core executing one high-level instruction every nanosecond.

Real Time Requirements

As the example we just discussed demonstrates, the nodes must be able to handle multiple tasks simultaneously. Especially in the case of audio, listeners are extremely critical to delays and gaps in audio streams. In many real-world applications, it is not enough to handle multiple tasks simultaneously, but they must be handled within tightly defined time slots as well.

Microprocessors have traditionally managed this through a combination of interrupts to inject the real-time element, and a system of round-robin processing of tasks wherein each task is processed for a certain period before the processor moves onto the next. As long as the processor is fast enough and the data input stream is slow enough, this approach gives the appearance of simultaneous task handling.

Over time, this has become more and more difficult, partly due to the ever increasing complexity of the tasks themselves and the growing tendency for these tasks to be multimedia in nature – i.e., sound and images.

Another issue, however, has been the direction of microprocessor design itself. Market pressures have moved those designs more and more in the direction of PC and server CPUs. And one of the basic weapons in the arsenal of the CPU designer has been that of cache memory. Much of the increased speed of today's microprocessors has been gained by larger and more efficient instruction and data caches designed to minimize the accesses to external memory or in the worst case, to slow-running disk drives. But caches are the enemy of real-time processing because they make the processor non-deterministic. Indeed, you cannot guarantee processing time of any given code segment because of the interference of the cache. Code executes at one speed when it's in cache, but the first time it is encountered it must be fetched from external memory, which adds significantly to the execution time. A similar issue occurs through the way modern processors try to predict the outcome of jumps and forks because

they add to the non-deterministic nature of the execution time.

Some processors attempt to solve the non-deterministic issue by incorporating two processors on the same chip with the idea that one would be dedicated to real-time processing while the other would handle operating system issues that are at the heart of cache problems. But this is frequently not enough, since as we've seen the computing node may be handling many tasks at once and you still have the issue of only having one processor to round-robin those tasks.

These issues can be resolved by simply putting dozens of core processors on each chip so that each of those tasks gets its own dedicated processor, or in some cases a half-dozen or more. Since each task is being handled by a dedicated processor(s) there is no longer the illusion of simultaneous processing – you have true simultaneous processing. This also solves the issue of interrupt latency. At the heart of all of these multi-tasking processor solutions is an interrupt clicking off time slots. As each interrupt occurs, the processor has to save the state of its registers and any task-related data in the process of being changed. The same thing happens in reverse when the task represented by the interrupt is completed, since the registers and data must be restored. The time to do that round trip is called interrupt latency, and it sets the minimum granularity of the real-time process… the minimum time that can be allotted to each task, even if there is no cache or predictive issue involved. A much simpler solution would be to have each task be handled by its own dedicated processor, thereby eliminating the need for interrupts and hence eliminating interrupt latency.

Local Memory Requirements

In simple, single-processor applications, we rarely think much about the memory other than to be sure there's enough. But as multiple processors are brought onto a single chip, the question of how to access memory comes into play. Generally, chips with several processor cores share some memory, either on-chip or external. Sometimes they even share a cache memory as well, which of course resurrects all of the non-deterministic issues associated with cache hits and misses. But even without the cache, if the only memory for the core processors is a common store on the chip, then there has to be some mechanism for arbitrating access to it as the processors fight to get data and instructions. That arbitration can be complex at best, and at worst can make the chip, once again, non-deterministic as well as creating a tremendous performance bottleneck.

This problem can be solved with the simple expedient of giving each core processor its own memory, both ROM and RAM, in sufficient quantity to enable most tasks to be executed totally from local core memory. No shared memory means no memory arbitration, full deterministic execution, and no performance bottleneck.

Low Power Requirements

Power requirements for mesh nodes vary greatly depending on the nature of the mesh and the application. Certainly in our example of the wireless home theater system, there's plenty of power available to run the node computer chip. But many of these mesh applications place the nodes remotely and require battery or even solar power to run them. That's certainly true of sensor driven mesh networks collecting data in the field, for instance. In some cases, the cost of the

power may exceed the cost of the node computer, so that anything to reduce the power consumption of the chip translates directly into cost savings.

Unfortunately high computing power is normally associated with high power dissipation. Simply put, the faster chips run the more power they dissipate during the charging and discharging of hundreds of thousands (or millions) of various parasitic capacitors associated with the transistors, the metalization, etc. This situation is at its worst when chips are designed with large, central clock trees where a majority of the nodes are driven synchronously.

This problem can be addressed in two ways. First, all of the core processors are totally asynchronous relative to each other – there is no central clock tree, or for that matter, no chip clock. Processors run as fast as native silicon allows, and they are naturally out-of-phase reducing the number of nodes being charged/discharged at any given instant. Second, the processors only run while they are doing work. That is, whenever they're waiting for data, either sending or receiving, they come to a total stop. There are literally no nodes in a waiting core processor that are being exercised, and since at any given instant, most of the core processors are in this waiting state, power is automatically reduced to the bare minimum, essentially just leakage current.

The Ideal Multicore Solution

By now we've already described many of the key points of the ideal multicore processor solution. Pack dozens of core processors on a single chip, each core with plenty of local RAM and ROM, and let them run asynchronously to increase speed and reduce power. Use these cores to address specific tasks so there is no interrupt latency or problem with trying to force a single processor to multitask. Make them very low cost, and the result is ideal for mesh networks in a wide variety of applications.

IntellaSys specializes in innovating multicore processor solutions that target embedded applications requiring low-power operation, fast operating speed and a small footprint. For more information visit: www.intellasys.net.